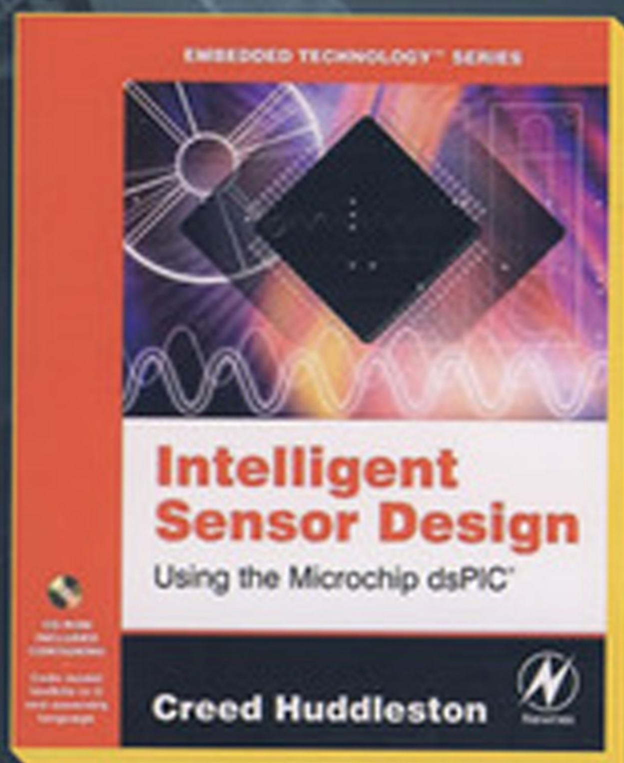


智能传感器设计

Intelligent Sensor Design

Using the Microchip dsPIC

[美] Creed Huddleston 著
张鼎 等译





人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵电子与电气工程丛书

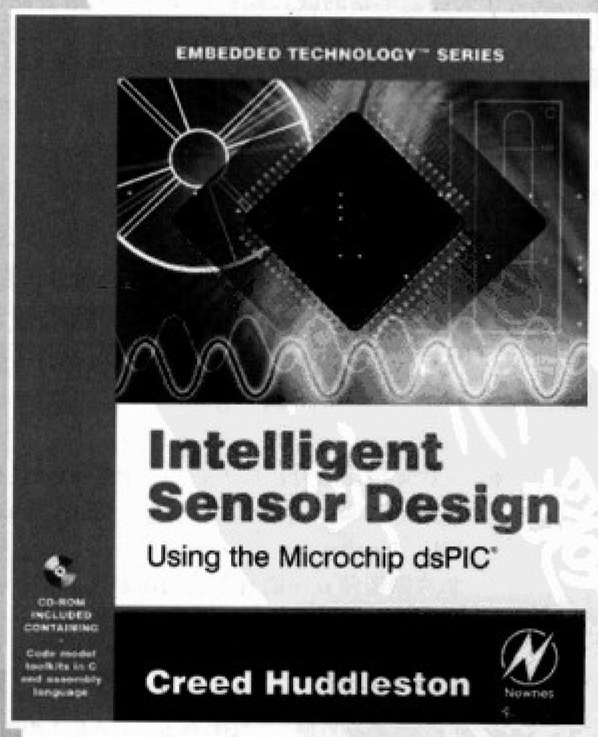
智能传感器设计

Intelligent Sensor Design

Using the Microchip dsPIC

[美] Creed Huddleston 著

张 鼎 等译



人民邮电出版社

北 京

图书在版编目（CIP）数据

智能传感器设计/（美）赫德尔斯顿（Huddleston, C.）著；张鼎等译. —北京：人民邮电出版社，2009.6
（图灵电子与电气工程丛书）
书名原文：Intelligent Sensor Design: Using the Microchip dsPIC
ISBN 978-7-115-20596-4

I. 智… II. ①赫… ②张… III. 传感器－系统设计 IV. TP212.6

中国版本图书馆CIP数据核字（2009）第044159号

内容提要

本书介绍智能传感器应用系统设计技术，力求理论与实践相结合，引导读者迅速掌握智能传感器的基础知识和设计方法，构建通用的软硬件开发平台，并轻松地开展应用设计与实验。主要内容包括智能传感器的基本概念，基于dsPIC系列数字信号控制器设计智能传感器的方法，以及温度检测、压力和称重检测、流量检测三种典型应用的设计实例。这些实例包含丰富的硬件电路图、软件流程图以及很多设计技巧，具有很强的参考性。

本书既可作为经验丰富的嵌入式系统设计师从事智能传感器系统设计的参考书，也可作为刚刚接触嵌入式系统设计的高年级本科生、研究生的学习参考书。任何有编程经验或者工程经验、又对智能传感器领域感兴趣的读者都会从中受益。

图灵电子与电气工程丛书
智能传感器设计

◆ 著　　[美] Creed Huddleston
译　　张　鼎　等
责任编辑　朱　巍

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：12.5
字数：337千字　　2009年6月第1版
印数：1-3 000册　　2009年6月北京第1次印刷

著作权合同登记号　图字：01-2008-3334号
ISBN 978-7-115-20596-4/TN

定价：45.00元

读者服务热线：(010) 51095186　印装质量热线：(010) 67129223
反盗版热线：(010) 67171154

译　者　序

传感器是人类了解世界的重要工具，它们能精确检测我们感兴趣的各种物理量，为控制和决策提供依据。然而在信息时代，人类已经难以独自处理如爆炸般涌现的信息了，我们希望传感器能够更加智能，比如能够独立完成信号采集、数据分析、结果报告以及信息共享等。因此，智能传感器应运而生。

智能传感器是一种新兴的、正在蓬勃发展的电子设备。它已广泛应用于生活、医疗及工业生产等领域，在温控系统、驾驶操控系统、重症监护系统、过程控制等应用中都有它们的身影。因此，智能传感器系统开发的市场前景广阔，这就要求大量嵌入式系统设计人员掌握智能传感器系统的设计开发方法。本书正是在此背景下出现的重要技术参考资料，它深入浅出地介绍了智能传感器的设计方法，其中既包含通用的数字信号处理技术，也涉及传感器信号检测与处理方面的技巧。借助于目前最流行的微处理器之一——Microchip公司的dsPIC系列数字信号控制器，读者可以在本书的指导下快速搭建智能传感器系统的软硬件开发平台，并开展应用实验。

本书兼具知识性和实用性。在内容组织方面，作者紧紧围绕智能传感器系统设计的特点展开论述，力求使读者能快速而准确地把握智能传感器系统的组成和特点。在语言描述上，作者没有使用深奥的理论，而是从物理意义出发，深入浅出地为读者展示出各种概念的直观含义，这些都体现出作者扎实的知识积累和丰富的实践经验。此外，书中的三个完整设计实例，能帮助读者解决智能传感器系统设计与实现中的很多实际问题，特别是附带资源①中的源代码和程序框架都已成功地应用于实际系统，因此具有很高的参考价值。

本书主要由张鼎翻译。此外，参与本书翻译的还有何小威、闫志强、邓彬、林龙信、肖枫涛、马蓉、陈钢、杨明军、焦贤龙、肖国尊等。张乐锋精心审阅了全书，提出了很多重要意见，对此深表谢意！Be Flying工作室（http://blog.csdn.net/be.Flying）负责人肖国尊负责本书译员确定、翻译质量和进度的控制，以及翻译思想的指导，在此予以衷心感谢。

① 本书附带资源请登录图灵网站（www.turingbook.com）免费注册下载。——编者注

谨以本书献给我亲爱的妻子Lisa以及我们三个可爱的孩子：Kate、Beth和Dan。和他们一起生活我很幸福。

本书还要献给我的姐姐Sarah，她爽朗的笑声令我永生难忘。或许，她会因为在这里看到自己的名字而惊讶不已（但愿她能喜欢）。

前　　言

任何书都不可能覆盖所有的主题，一本书也不可能适用于所有的读者。尽管我非常希望本书能适合所有的读者，但它还是没有例外。本书主要面向以下三类读者。

（1）经验丰富的嵌入式系统设计师。他们希望尽快熟悉Microchip公司的dsPIC系列DSC，并且没有足够的时间通过研究大量详细文档来熟悉和掌握系统。

（2）刚刚接触嵌入式系统的工科学生。他们希望能得到额外的指导以掌握如何将系统整合起来。

（3）任何有编程经验或者工程经验又有兴趣学习更多智能传感器领域知识的读者。

为了能读懂本书的大部分内容，读者应当熟悉基本的C语言编程。书中的实例都未使用难以理解的语句结构。本书的重点在于应用，特别是在传感器系统的设计方面，而不是在编程语言本身。如果读者不熟悉C语言，那么建议读者先尽快掌握它，这方面有很多优秀的书籍可以参考。

这也不是一本数字信号处理（DSP）方面的入门书籍。尽管书中简明扼要地（只花费一章的篇幅）介绍了DSP的基本概念，但这些并不是严格的理论阐述。这也并不意味着可以轻视类似这样的图书。相反，这些书非常有价值，多年来我已经买了很多这样的书！不过，在本书中我们将DSP视为一种工具，并且假设读者已经很熟悉该工具或者能够自学掌握。由于本书的目的只是使读者能够直观地理解必要的DSP理论，因此更加详细的阐述只会浪费时间。况且，已经有很多比本书作者更权威的人士撰写了这方面的书籍。

本书也不是一本硬件设计手册。尽管我们在很多实例中都讨论到电路方面的内容，但重点是建立基本的信号调理平台，并通过dsPIC系列DSC软件的功能从被监控系统中提取有用信息。和编程以及数字信号处理一样，硬件设计领域的专家们也已经撰写了大量优秀的论文来研究模拟和数字电路设计中错综复杂的问题。

前面说了这么多本书“不介绍什么”，下面谈本书到底介绍了什么。本书的目标是使读者能够迅速搭建软硬件开发平台，学习必备的知识，以便轻松地利用各种设计智能传感器的方法进行实验。智能传感器是一种新型的传感器，它具备更好的测量能力以及传统传感器所没有的特性，并且能轻松地接入各种监测与控制系统，因而迅速取代了传统的传感器。本书介绍了智能传感器在温度测量、称重监测以及流量检测领域的三个完整应用实例。附属资源中有这些应用的源程序，包括详细的注释，此外，附属资源还有各种有用的因特网资源链接。

有一点需要强调，作者在本书中采用了Microchip公司的很多软硬件开发工具，这使得本书乍看起来像是给Microchip公司做广告。然而事实绝非如此！本书从未得到来自Microchip公司的任何资金资助。尽管作者本人所在的Omnisys公司是Microchip公司的授权咨询商，但是Omnisys公司同时还是包括Cypress、Freescale以及Lattice公司在内的许多其他半导体公司的授权咨询商。

之所以采用Microchip公司的开发工具，是因为它的供货方便、价格低廉（经常有免费的），并且提供了用户论坛，用户遇到难题时可以获得帮助。例如，Microchip C编译器（学生版）以及MPLAB集成开发环境（IDE）都可以从Microchip公司的网站上免费下载，而用于设计模拟抗混叠滤波器（以后还会支持设计更多种类的滤波器）的软件Filter Lab 2.0也可以免费下载。此外，Microchip公司还免费提供dsPICWorks软件（这是一款DSP分析工具），而dsPIC Filter Design软件的价格也很容易接受。作者所认识的大多数开发人员都不愿意为了实验自己的一些想法而花费大量资金来购置开发工具，而Microchip公司正好提供了价格低廉或者免费的设计工具。作者本人在日常开发工作中就使用这些工具，因此对它们很熟悉。

希望读者能像我愉快地撰写本书一样愉快地阅读本书。我从事嵌入式系统，特别是硬件实时控制和通信方面的工作已经有20多年了。在这段时间里，我有幸和很多非常聪明而又见解独到的技术人员共同解决了很多难题。如果我圆满地完成了写作任务，那么亲爱的读者也一定会从中受益的。

目　录

第1章 智能传感器概述

在如今这个即时获取信息的年代，人们要求或者希望在他们需要信息的时候就能以所需的格式和能够承受的价格（最好是免费）得到它们。近百年来最伟大的管理学大师彼得·德鲁克（Peter Drucker）曾指出，信息和其他有形产品不同，它们并不符合经济学的短缺理论（经济学认为“物以稀为贵”）；相反，信息的数量越多、传播越广，其价值就越大①。很多理解该概念的个人和组织已经开始挖掘在全球的商业企业、学术机构以及非盈利组织中潜藏的巨大价值。他们将海量的原始信号数字化，经过分析后得到有用信息，然后通过标准通信链路将信息发送至组织内外的其他人，以便完成重要的工作。

这种新的方式给社会的经济活动、智力活动以及人们的日常生活带来了巨大的影响。人们现在常说，如今是工作在因特网时代，其中的时间和空间都已被大大压缩。我们可以将全球各地的信息迅速而可靠地分发至其他任何地方，于是印度的班加罗尔到纽约的距离就和波士顿到纽约的一样近。在这个崭新的世界里，人们的工作方式也与以前截然不同，每个人或团体都能和遍布偏远地区或世界各地的人们结成团队，共同提出新思想、设计新产品或新服务，可以为一部分人创造惊人的价值，也可以破坏另一部分人的生活。事实上，这种被作家托马斯·弗里德曼称为地球“扁平化”的新模式，体现出人类看待世界以及交流方式的巨大变化。

1

有意思的是，在平凡的传感器世界正在发生一种几乎相同却不为人知的巨变。对于外行而言，*传感器*（*或称作敏感元件*）是一种利用传感器自身物理特性测量用户感兴趣的物理量的设备。这是对传感器概念的笼统解释，即通过监测一个相对容易观测的参数的变化，来推断出另一个难以观测的参数的大小。

例如，球泡型水银温度计是一种大家熟知的非电子型温度传感器（见图1-1），其中水银的容积能根据外界温度的变化而发生收缩或者膨胀。在这里，我们需要测量的物理量是温度计所插入的物体的温度，而我们直接监测的传感器的固有物理属性则是温度计中水银柱的高度。

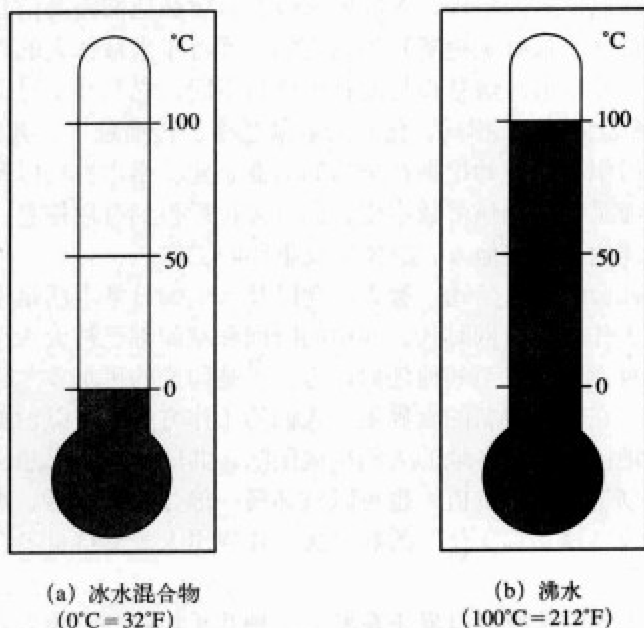
那么我们到底能用传感器测量哪些类型的参数呢？答案是种类很多。事实上，只要是我们能想到的，就可以测量。最常见的测量参数可能是温度，此外，还有压力、加速度、湿度、位置、pH值等数千种。传感器的用途之所以如此广泛，不仅是因为它们能够精确测量各种参数，而且还在于传感器能够在人无法接近的环境下完成测量。无论是测量高炉（blast furnace）中心处熔化的钢的温度，还是检测海洋中数千英尺深处的洋流，传感器总能提供精确的信息，从而使人们能够监测并且控制各种重要过程。

2

乍一看，传感器就像是一件舒适的运动衫，你可能会因为拥有它而感到高兴，但不会特别兴奋。要是读者产生这样的第一印象那可能就完全错了。事实上，截至2005年，全球约有

① *Management Challenges for the 21st Century*, Peter, F. Drucker. Collins, 2001。

64亿人口[①]。碰巧的是，2005年美国的工业传感器市场大约为64亿美元[②]，而全球市场则高达400亿美元。可见，世界上的传感器数量比人口数量多得多。从日常琐事到科学的最前沿都要求使用传感器，而且人们也乐意花钱以获得传感器带来的好处。传感器领域蕴含巨大利益，它集市场需求、技术挑战以及经济机遇于一体。竞争促使传感器世界出现一种突破性的器件，这就是智能传感器。



(a) 冰水混合物 (0℃=32℉)　　(b) 沸水 (100℃=212℉)

图1-1 两个球泡型水银温度计指示被测物（冰水和沸水）在不同条件下的温度

那么什么是智能传感器呢？从概念上讲，它是一种新的电子传感器件。它真正完全改变了我们采集周围数据的方式、从数据中提取有用信息的方式以及利用新的信息执行各种操作的方式，并且比以前更快速、更精确、更安全而且成本更低。更妙的是，我们可以将单个智能传感器检测到的信息传输至其他智能传感器或者其他系统，产生杠杆效应，从而实现以前无法完成的任务，并且给各种应用系统的性能带来难以置信的提高。这些话是不是听着有些耳熟呢？

1.1 常规传感器并非完美

在探讨智能传感器之前，我们首先应当更加深入地研究一下常规传感器，这会为我们后面理解智能传感器打下坚实的基础。尽管常规传感器已经很不错了，但是大多数传感器在技术方面或者经济性方面都有些缺点。为了确保传感器能够有效地测量，通常必须对它们进行标定，也就是说，传感器的输出必须符合某个预定的标准，这样它们测得的数值才能真实地

① 根据*CIA World Factbook*，截至2005年7月，全球总人口估计已达6 446 131 400。http://www.cia.gov/publications/factbook/rankorder/2119rank.html。

② 根据B. L. Gupta为Business Communications公司所做的研究（GB-200N 工业传感器技术与市场），2004年美国的工业传感器市场的规模达61亿美元，预期的年增长率达4.6%。http://www. Bccresearch.com/instrum/GB200N.html。

反映被测参数的值。在球泡型温度计的例子中，就必须标定水银柱旁的刻度，从而使水银柱的高度对应一定的温度。如果传感器未被标定，那么它们所报告的数值就不准确，使用这样的信息会给系统带来大麻烦。然而，并非所有的情况都需要达到相同等级的准确度。比如，即使你房间内的温度计有1～2°C的误差，也不会造成明显的影响；因为你只是想将室温调节到使你舒适的温度而已。但是，如果在化学反应过程中同样出现温度偏差，那就可能使所得的化合物发生明显变化，生产出一批无效产品，严重时甚至会造成爆炸！后面我们还将深入探讨有关标定的话题，但是读者从现在开始就应当建立这样的概念：能够准确标定传感器是一种必备的优良特性。和标定传感器一样重要的另一个概念是，传感器一旦被用于现场，往往就很难甚至不能再对其进行手工标定。

3

使用传感器需要注意的第二个问题是，传感器的性能会随时间发生变化，这种现象被称为*漂移*。例如，假设要通过测量电阻两端的电压来检测电路某部分的直流电流（见图1-2）。

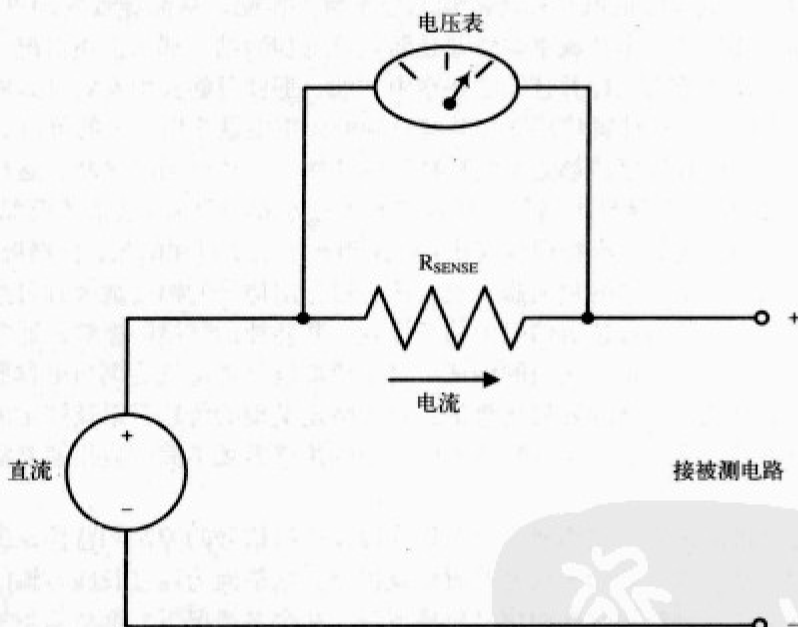


图1-2 使用阻性传感器件测量电流的例子

4

本例所采用的敏感元件是电阻，而被测物理量是电阻两端的电压。由欧姆定律[①]可知，电阻两端的电压会随着流过电阻的电流值的变化而变化。随着电阻的老化，其化学特性就会发生变化，从而使电阻的大小随之改变。例如，对于新系统，当电阻上流过100mA电流时，所测得的电压为2.7V，而5年后在同样大小的电流条件下，所测得的电压就可能是2.76V。尽管这0.06V看起来不大，但是对于某些应用就会造成严重影响。和标定一样，有些情况比其他情况对漂移的容限要求更严格，除非我们通过某种方法来补偿漂移，否则传感器的性能就会随时间而发生变化，而这些变化是我们不希望出现的。

顺便说一句，你知道在上面的例子中，为什么用电阻来充当敏感元件，而不是用电压表来测量电压？尽管探讨这个问题看起来有点儿学究气，但是这的确很重要。前面已经说过，

① 欧姆定律是$V=I*R$，其中V是电阻两端测得的电压（单位：伏特），I是流过电阻的电流（单位：安培），R是电阻的阻值（单位：欧姆）。欧姆定律适用于纯阻性元件，这是在本例中需要注意的。

在这个例子中，我们是通过测量电阻两端的电压来检测电路中的电流的。因此电阻才是主要敏感元件，而其两端的电压只是随我们所关心的参数而变化。电压表是我们测量所关心的参数时使用的次要敏感元件。也许有人会说，电压表本身也有标定和漂移的问题。区分主要敏感器件和次要敏感器件很重要，原因是这决定了我们所测数据的精度。如果没有透彻理解被测参数，那么就无法构建一个真正能够测量我们所需要信息的系统，或者会引入其他问题而影响测量的准确度。我们还会在后文中专门讨论一些与此相关的特殊问题。

第三个问题是，不但传感器本身的性能会随时间而变化，就连它们所处的工作环境也会发生变化。有一个很好的例子可以说明这个问题，那就是内燃机的电子点火系统。当系统刚刚调试完毕时，传动带是绷紧的，火花塞是崭新的，喷油嘴和空气滤清器也都是干净的。但从那时起，情况就越来越差：传送带变得越来越松，火花塞和喷油嘴上都粘有沉淀物，空气滤清器也被越来越多的污物和灰尘所阻塞。除非电子点火系统能够检测出这些变化并做适当调整，否则点火设置、点火时机就与发动机状态越来越不匹配，从而导致发动机性能下降、燃油效率降低。如果你只是开车在城里转悠并且随处可见加油站，那么上述情况可能对你影响不大，但是如果你是在跨海飞行并且无法在空中加油，那就可能会引发灾难。能够补偿工作环境中发生的巨大变化，这使得传感器在某些特别的应用中显示出巨大的价值。

5

第四个问题是，大多数传感器都需要某种特殊的硬件——*信号调理电路*，这样才能用于监测或者控制的应用中。信号调理电路能将所监控的传感器的物理量（通常是模拟电压信号，它按某种预先计划好的方式随被测物理量变化）转换为其他系统可用的量。根据应用的需要，信号调理电路可能只是一个简单的放大器，它将传感器输出信号的幅度放大到可用的水平；也可能是复杂电路，它能滤除传感器信号中的干扰量，并能补偿环境的影响。通常，信号调理电路需要针对所用的传感器进行适当的调试，对于模拟信号来说就是调节电位器或者其他微调器件。此外，信号调理电路的发展趋势是：对于特定类型的传感器以及特定的应用本身来说，信号调理电路都是唯一的。这意味着不同类型的传感器或不同的应用通常需要采用专用电路。

最后，常规传感器通常需要在物理空间上靠近接收检测信号的控制和监控系统。一般来说，传感器距离监控系统越远，检测信号的质量就越差。这是因为通过长线传输的传感器信号对电噪声很敏感，从而导致接收端的信号质量下降。在很多情况下，都要采用特殊的（昂贵的）电缆将传感器与监控系统相连。电缆越长，安装的成本也越高，这可能使最终用户难以承受。另一个与此相关的问题是，多个系统之间难以共享传感器检测信号，特别是当这些系统彼此远离时更加困难。无法共享传感器的输出信号看起来无关紧要，但是这会严重限制系统的安装规模，从而不得不花费更多的资金来安装多个冗余的传感器。

我们真正需要做的是研究新技术，解决或者至少能大幅缓解传感器的标定、漂移以及信号调理等方面的问题。如果我们能找出一种简便地共享传感器输出信号的方法，那么我们就能够解决安装规模的问题。下面将首先介绍在这方面目前已经做出的成果，然后研究这些新方法对传感器世界的影响。

6

尽管传感器种类繁多（包括电子的、机械的、化学的、光学的等），但是本书着重研究电子传感器，这是因为一个简单而强有力的理由，那就是我们可以很容易地将电子传感器的输出与计算元件（通常是微处理器）相连。这样，我们就能利用这些计算元件为传感器增加智能。后面将会看到，这些新增的智能极为有用。

1.2 第一步——传感器信号数字化

当工程师设计带有传感器的系统时，他们首先会建立传感器对被测物理量的响应的数学模型，以及信号调理电路对传感器输出信号的期望响应的数学模型，然后用电路实现这些数学模型。尽管上述模型都很好，但是必须注意，这些模型只能近似等于实际电路的响应（尽管通常希望这些模型都是精确无误的）。如果能以数学形式表示系统中尽可能多的部分，那么模型就更接近于实际响应。这是因为，数字不会随时间而改变，并且也可准确地和轻松地处理它们。事实上，这种*数字信号处理*（DSP）的原则已经确立并得到广泛应用，DSP就是利用数学而不是电路对信号进行处理的。利用DSP可以很容易地实现标准变换，比如通过滤波去除不想要的噪声或者通过频率映射来识别特定的信号分量。此外，利用DSP理论还能实现一些即使是最先进的电路也无法完成的处理。

正因为上述原因，如今的设计中都包含一级信号调理电路，它负责将模拟信号转换为数字量。这一步被称作*模－数转换*，或称A/D转换，也可简记为ADC。这一步极为关键。因为一旦将传感器信号转换为数字量，那么就能通过微处理器中运行的软件来完成信息处理。通常所说的*模－数转换器*其实是一枚单片的半导体器件，它能够精确地、稳定地工作于变化的环境中。由于大多数环境补偿电路都可以集成到ADC器件中，而滤波处理则可由软件实现，因此，对信号调理电路的需求会显著减少。我们很快就会看到，由于系统的器件大大减少并且所有处理几乎都通过数学运算完成的，因此无论是从系统性能还是商业前景的角度来看，都是大有益处的。

1.3 第二步——增加一些智能

将传感器信号数字化之后，主要有两种途径来处理这些数据，以实现算法定义的功能。第一种是采用专用的数字硬件电路，通过硬件连线实现我们设计的算法；另一种则是采用微处理器完成运算。一般来说，专用硬件比微处理器系统的速度更快，但是其成本较高并且缺乏灵活性。当不需要专用硬件那么高的运算速度时，由于微处理器适用的场合更广泛，因此具有更好的设计灵活性，并且可能成本更低。

一旦系统具有了智能，就能解决我们之前提出的问题。比如可以自动完成标定，也可以通过纯数学的处理算法消除器件漂移，甚至还能通过定期监控环境条件并自动进行适当的调整来补偿环境变化。可见，只要赋予系统一个大脑，就能使设计师的生活自在得多。

第3章将会看到，有一种相对较新的微处理器，即所谓的*数字信号控制器*（digital signal controller，DSC），它们正被迅速应用于那些要求低成本、高集成度的产品中（即将多种功能集成在一枚半导体芯片上），并且能有效地运行分支密集软件和运算密集软件[①]。尽管DSC通常难以达到专用数字处理硬件那么高的速度，但是它们能满足绝大多数应用场合的速度要求，并能实现必要的算法。目前这才是最关键的问题。

1.4 第三步——实现快速而可靠的通信

最后还剩下一个问题：如果能实现传感器数据共享，那么就可以很容易地扩大那些需要

① 分支密集软件是一种在执行指令时频繁变化（所谓的*跳转*）的软件。运算密集软件是一种花费大部分处理时间来专门执行数学运算的软件。

共享传感器输出信号的系统的规模。这里再次强调，由于传感器的输出信号是数字的，因此能够可靠地实现共享。就像在人类的世界中共享信息后会使信息量增大一样，与本系统或者其他系统中的某个部件共享传感器测量值后也会使系统的总测量值变多。为了实现这个目标，我们必须装备带有标准通信接口的智能传感器，使它们能与其他部件交换信息。通过标准通信方式，我们就能保证尽可能广泛、简便和可靠地共享传感器输出信号，从而最大限度地发挥传感器及其产生的信息的作用。

8

1.5 第四步——将所有的东西连在一起，就得到一个智能传感器

这里要强调大多数工程师认为的智能传感器（有时也称作*灵巧传感器*）的三个必备特征。

(1) 含有用于测量一个或者多个物理量的*敏感元件*（本质上就是我们提到过的传统传感器）。

(2) 具有用于分析敏感元件所测结果的*运算元件*。

(3) 与外界相连的*通信接口*，它使传感器能在一个更大的系统中与其他部件交换信息。

其中，后两点是智能传感器与常见的常规传感器的主要区别（见图1-3）。智能传感器具有将数据直接转变为信息的能力，能在本地使用信息，还能将信息传输至系统中的其他部件。这些都是常见的常规传感器无法做到的。

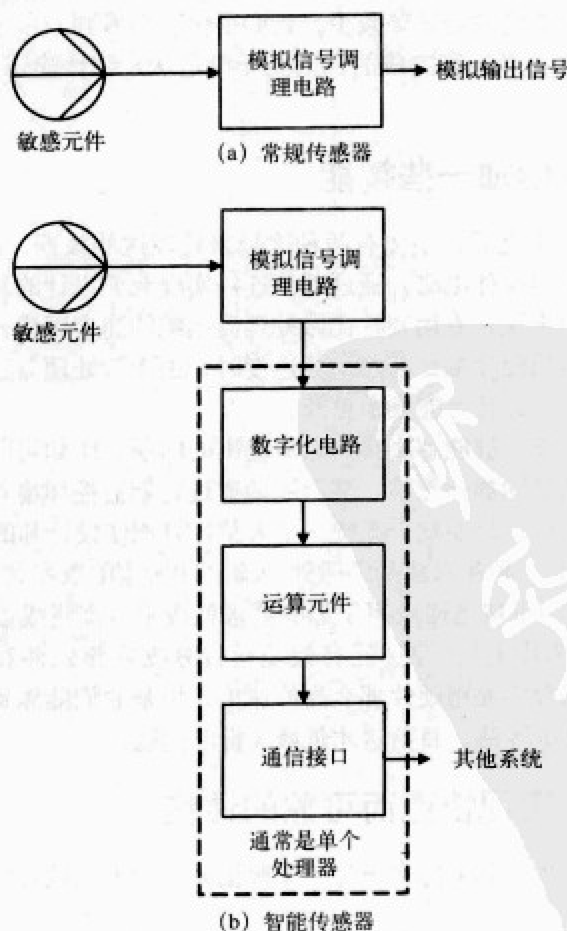


图1-3 常规传感器和智能传感器的框图

本质上看，智能传感器使传感器世界“扁平化”，它使得传感器能与附近的或者遍布全球的其他传感器相连，共同完成以前无法完成的任务。更为重要的是，由于智能传感器的大多数功能都由软件实现，因此设计师只需改变传感器的软件配置就能得到各种各样的产品。

这对智能传感器供应商造成两点重要影响。一个是供应商要从提供硬件产品转变为提供软件产品。尽管传感器仍然需要一个基本的硬件平台（毕竟它还是个物理设备），但是提高或争夺利润的主要因素已不再是硬件，而是控制智能传感器的软件。由于制造商只需稍改一点儿软件就能添减产品的功能，因此他们几乎可以即时地改变产品的搭配。那些定制的产品甚至可以直到完成最终测试和运输前才进行配置。此外，一种硬件平台可以用于不同市场、不同价格的多种产品。最后，一旦供应商开发出了新功能，那么无需增加产品的硬件成本就能使之具备新功能，从而使边际利润猛增。

另一个影响是，由于智能传感器能与外界相连，因此供应商如今可以在调试现场收集传感器的运行信息，并能在传感器出厂后更新设备的软件。传感器工作现场的信息不但能使传感器制造商准确掌握用户需求及其关注点，而且还能提供——对用户来说最为重要的——定位故障或缺陷的可靠数据（很可能就是用户将来最在意的东西）。掌握这些信息后，传感器制造商就能快速地为产品增加新功能、根据用户需求提供相应的配置并实施产品维护，而且在进行上述工作时都无需接触到传感器本身。如今，供应商可以足不出户地提供低成本的服务，这又给供应商带来了额外的价值和利益。下面以*Harvard Business Review*杂志中的一篇报道[①]作为例子，来说明这种变化。

由于受到来自当地竞争者的产品价格以及利益的压力，大多数产品制造商都无法应付比每小时90～110美元还高的技术支持费用。但是，GE能源公司通过利用有效的基于网络的远程服务，就可以为技术人员支付每小时500～600美元的报酬。更为重要的是，由联网的不间断监控系统得到的这些信息还为GE公司提供了更多的商机，比如掌握客户的备件库存或者为客户和GE公司提供服务，此外还支持个人获取有关设备状态的规范的数据和知识。

1.6 智能传感器局限

既然将标准的、独立的传感器升级成互联的智能传感器后可以获得巨大效益，那么为何不将所有的传感器都做成智能的呢？原因其实很简单，因为有些情况并不适合采用我们前面介绍的互联的智能传感器，我们必须认识到这一点。一般说来，在以下情况中为传感器增加智能并不合适。

- 无法在合理的时间内从用户身上赚回产品开发和制造的成本。
- 终端用户无法或者不愿意提供智能设备运行或与之通信所需的基础条件。
- 某些应用的环境约束不允许增加实现智能和互联所需的电路。

1.6.1 开发和生产成本超过从用户处可获得的利润

为了使产品具有较长的使用寿命，用户通常愿意为此支付必要的设备开发与制造费用。这个原则不但适用于像纸巾这样的普通货物，也适用于采用高新技术的产品（比如智能传感器）。没有哪家公司愿意长期生产一种制造成本比利润还高的产品。因此，在投入时间和资

① *Four Strategies for the Age of Smart Devices*, Glen Allmendinger and Ralph Lombreglia. *Harvard Business Review*, October, 2005. Reprint R0510J。

9

10

11 金为设备增添智能之前，传感器制造商应当确定它们的用户是否愿意接受产品差价或为更优质的服务买单，这部分费用至少也要抵消开发成本以及增加的生产费用。（如果能降低成本，那么制造商就可能愿意采用新设计。）除非用户充分重视智能设备带来的利益，否则制造商最好还是继续生产非智能产品。

起初，可能用户能清楚地看到增加智能带来的利益。但是，有些应用完全是由成本决定的，利润很低，因此这类用户根本不愿意为新技术投资。比如低档的一次性塑料餐具，制造商从每个产品上仅能得到几分钱的利润。像这种利润极小的情况，生产商肯定不会为设备投资很多，他们是否愿意购买设备只取决于产品销售价的底线，任何需要额外付钱的东西在他们看来都是毫无价值的。

1.6.2 缺少必需的基础条件

第二种不应该或无法使用智能传感器的情况是用户无法提供智能传感器工作所需的最低基本条件，即传感器的供电要求和通信信道要求。与前面提到的传感器制造商要能赚回生产成本的条件类似，这里的情况是制造商可能无法以较低成本来增加智能传感器工作所需的额外条件。供电功率和通信信道都是保证智能传感器能够正常工作而必不可少的基础条件。没有供电，传感器就可能根本无法开机；没有通信信道，传感器就无法输出采集到的信息。

有关智能传感器实现方面的问题也不可忽视。虽然越来越多的制造工厂都已建立了数字化网络，但是这会增加制造商的运营成本，成为减少利润的致命杀手。尽管一些新的网络协议可以通过导线在为设备供电的同时还能用于通信，例如以太网供电技术（Power-over-Ethernet，PoE），但是那些老工厂的联网成本会特别大。目前，市场上出现了新一代的低功耗型无线传感器，它们能够解决上述问题。遗憾的是，尽管无线传感器的长期运行成本可能

12 很低，但是其初始购置设备的花费则可能比有线网络更大。

1.6.3 环境条件不允许增加电路

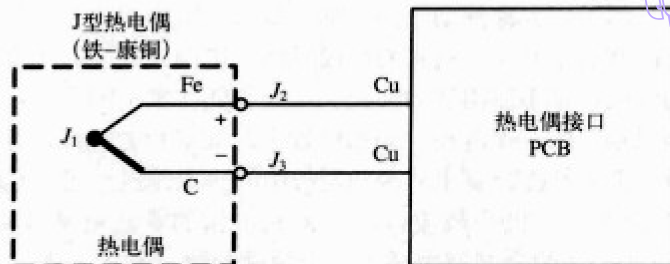
最后，也是最少遇到的无法使用智能传感器的情况是，某些特殊应用的环境条件不允许增加额外的电路。这些环境条件可能是设备尺寸、工作温度、严重的振动或者是暴露在腐蚀性化学物中。在上述情况下，尽管通过尽早地将所测参数数字化可以显著提高传感器的性能，但是，由于受到工作条件的限制，采用加固型常规传感器可能是唯一的选择。

1.7 智能传感器实例

结束本章之前，我们来介绍三个智能传感器的应用实例，其中有两个实例来自工业过程控制领域，另一个来自车辆控制领域。

1.7.1 多通道数字温度传感器

温度是各种工业过程控制中常用的控制参数，热电偶则是最常见的温度传感器之一。从某种程度上看，热电偶是一种极为简单的敏感元件，它是两种不同金属在一个点构成的金属结。根据塞贝克效应（Seebeck Effect）可知，连在一起的两种金属之间会产生电压，并且电压的大小随金属结的温度的变化而改变，见图1-4。



注意图中不同的金属其实构成了三个节点，因此实际上电路中有三个热电偶

J_1——热电偶传感器（唯一期望存在的热电偶）

J_2——铁引脚与PCB的铜引线构成的结

J_3——康铜引脚与PCB的铜引线构成的结

图1-4 基本热电偶的简图

13

在后面的一章中我们还将详细研究热电偶，但在这里需要读者了解的重要概念是热电偶的输出电压非常小，通常是毫伏级。并且，它往往工作在高达数百伏的电气噪声环境中。更为复杂的是热电偶的温度响应是非线性的，因此必须在使用所读取的温度值之前对其进行线性化处理。此外，使用热电偶还有其他一系列重要问题。但是，从下面两个例子中足可以看出智能传感器如何在注模机这样的极恶劣环境中实现准确的温度测量。

可能有些读者还不熟悉注模过程。注模过程是先将固态的塑料片加热到300～900°F[①]使之融化，接着在高压下（可达10000～30000PPsi[②]）将融化的塑料注入到模具中，然后使塑料冷却，就能使之恢复成和模具形状相同的固态。如果重复上述过程就可以快速生产出部件。注模生产成功的关键就是缩短生产周期（每次生产时从打开至闭合模具之间的时间）并降低废品率。只要模具能够以一定的利润率生产出高质量的部件，那么生产过程其实就像“印钞票”一样，并且赚钱的速度就等于单位部件的生产周期。此外，注模生产的另一个关键问题就是如何保持注模机中的塑料各点的温度基本相同，这就要求温度传感器（热电偶）分布在关键点上。

使用热电偶的缺点就是用于制造热电偶的金属连线很昂贵。一般的注模机尺寸都不小，因此它通常包括多套温度监视与控制区（大型系统中可能包含250个区域，甚至更多）。因此热电偶的引线必须很长才能与温度控制器相连，这就可能导致出现最糟糕的情况：不得不将昂贵的多股导线连得很长！一家温度控制领域的龙头公司已经意识到，通过在模具中对温度读数进行数字化，然后通过标准铜电缆（便宜得多）将数字化的读数传输到控制器，可以为用户节省大笔资金。此外，由于热电偶读数的变化速度相对较慢，因此可以按照时分复用的方式将数字化读数发送到控制器，这样上述方法还能处理多通道的热电偶读数。于是，一个设备就可能接收多达96个通道的热电偶数据，从而能将一个带有长距离热电偶引线的造价昂贵、易受噪声影响的系统变成了容易处理的、只有一对铜导线的系统。

14

1.7.2 流量传感器

尽管塑料在被制成模具前必须始终呈融化态，但是一旦达到了期望的形状，就要立即将塑料定型为固态。为此，模具中应设置冷却通道，并在其中通有循环冷却水或其他液体，以

① 华氏度和摄氏度的换算关系：设华氏度为t，那么摄氏度为 $\frac{5}{9}(t-32)$。——编者注

② 1PPsi=6.894 76 × 10³Pa。——编者注

便快速地排放塑料上的热量。如果流动不畅，那么冷却液就会因为长时间与高温模具接触而升温。这样就会降低冷却效率并延长模具的凝固时间，进而延长生产周期并损失利润。这显然是任何设计合理的注塑机都不应出现的情况，因此智能注塑机的冷却系统中配有流量传感器，以确保冷却液能以期望的速度流动。如果已知冷却液本身的特性，并测得冷却液在不同两点的温差以及流量，那么就能计算出从冷却系统中的一点传递到另一点的BTU值[1]。

流量传感器有多种形式，但是最流行的一种是所谓的*在线流量传感器*（in-line flow sensor）。这种传感器在流动的冷却液中插入一个像螺旋桨一样的设备，它能指示出流过传感器的液体的流量。但是，螺旋桨上的轴承会与冷却液之间产生摩擦，并导致螺旋桨出现偏心移动，从而使流量读数下降。有一家技术领先的公司已研制了便携式手持流量传感器，它采用了特殊的滤波算法，能够补偿螺旋桨轴承的摩擦力，并且长期使用都也能保持读数准确，从而显著地延长了传感器的使用寿命。

1.7.3　线控驾驶转向位置传感器

最后一个智能传感器的实例来源于车辆控制领域，特别是船舶上的线控驾驶系统。在普通的机械式操控系统中，方向盘和转向操控界面（比如汽车的轮子或者是船舶的桨片）之间采用物理连接。当驾驶员转动方向盘时，产生的转动量会通过一系列的机械连接传递至操控界面，并使之产生对应的变化。根据系统的配置情况，驾驶员从驾驶系统（道路或者水面）得到反馈，它能帮助驾驶员调节动作。尽管机械式驾驶系统很可靠，但是它必须配备昂贵而且极其复杂的机械机构才能控制多个方向盘。汽车里是不会出现这种情况的，它通常只需要一个方向盘，但是对于大型轮船就面临这个问题，它通常需要两个方向盘，一个位于船头，在停靠码头时使用；另一个位于船尾，在平时巡航时使用。

15

相反，在线控操纵系统中，大多数的连接由电控设备代替，尽管驾驶员只转动了方向盘，但是方向盘是通过电子线路而非机械机构连到操控界面的。线控操纵系统不仅能够显著降低驾驶系统的体积和重量，而且还有很多优点。此外，这种操控系统还能方便地装备两个或者多个方向盘，只要用电线连接上就可以了，而无需其他额外的机械连接。但是线控驾驶系统也有缺点，那就是驾驶员无法从操控界面获得反馈信息，这可能会使驾驶员因为自己的动作和船舶的反应脱节而感到不适。幸运的是，有一种特殊材料改变了这种没有反馈的情况，这种材料能根据通过它的磁场的强度来改变自身的特性。由此设计出了一种转向位置传感器，它能在每秒内频繁地检测方向盘的位置，这样先进的线控系统就能根据驾驶员转动方向盘的速度等检测量来调节该材料的密度，并作为反馈信号传递给驾驶员。此外，传递给驾驶员的反馈信号还能根据操控界面的条件而改变，从而让驾驶员享受舒适而安全的驾驶体验。

1.8　本书后续内容概述

到目前为止，我们已经初步了解了智能传感器是什么，并看到它们是如何改变世界的，此外还看到智能传感器的一些应用实例，在本章最后，将概述本书的后续内容。

尽管本书的各章内容基本上是彼此独立的，但是作者仍然建议读者先阅读第2章，然后再读第3章。理由是，在第3章全面介绍dsPIC系列DSC的功能时需要用到第2章讨论的一些概念。第2章，我们将探讨数字信号（DSP）处理原理的直观理解，这是后续几章实现三类智

① BTU，British Thermal Unit，英制热量单位，1BTU=1055.06 J。——译者注

能传感器的基础。由于市面上有很多有关数字信号处理方面的优秀书籍，它们比本书讲述的深入得多，因此，本书并不指望读者通过阅读这一章内容就立即成为DSP方面的专家。相反地，本章只是希望读者能够理解为什么要使用某项DSP原理，以便他们能够牢固掌握何时使用特定方法以及在什么情况下另一种方法的成功概率更大。 16

在第2章介绍DSP基本原理的基础上，第3章将深入剖析dsPIC系列DSC，这是一类特别适用于实现智能传感器的数字信号控制器。我们将集中介绍其中的一款芯片——dsPIC30F6014A，它具备dsPIC系列芯片的很多常见功能。当然，我们还将介绍dsPIC系列DSC的软件开发环境，以及Microchip公司提供的很多软件工具和程序库，它们都有助于提高基于dsPIC系列芯片的产品开发速度。在第3章结尾，我们还将探讨一个基本的传感器软件架构，它将在本书后面几章中用于研制各种智能传感器。

第4章将通过研究有关处理器与其他设备间的通信问题来进一步掌握dsPIC设备。有些设备可能是增强系统功能的片外板上外围元件，或者可能完全是另一个系统，并且距离传感器单元本身很远。或许有人会认识到，上述两种不同的情况可能需要采用完全不同的通信技术。幸运的是，dsPIC系列芯片配备了多种通信接口，完全能够满足这种要求。特别是，我们后面会用到它的RS-232/485总线、串型外围接口（SPI）总线以及控制区域网络（CAN）总线。这些都是工业标准通信接口，因此能使基于dsPIC芯片设计的设备在各种工作条件下与多种系统接口。特别是，CAN总线具有在同一网络中不同节点间实现可靠的中速（最快1Mbit/s）通信能力，因此已被大量用于工业产品及车辆中。鉴于通信协议繁多，因此有必要从中筛选出那些重要的来介绍，以节省读者的时间和精力，并避免走弯路。

第5章将接着第3章、第4章讨论如何使用一套模块化的软件工具箱，并基于dsPIC处理器完成各种任务。将这些软件工具集中到一起使用，就能在第3章介绍的传感器软件框架上实现一个多通道滤波器，利用这个滤波器，我们能够可靠而快速地对多通道的信号实现滤波（去除噪声）。 17

第6章至第8章将深入研究智能传感器的三种典型应用，包括温度检测、压力检测和流量检测。这部分内容将针对每个应用的要求，基于第3章、第4章开发的传感器软件框架以及第5章开发的DSP处理模块共同实现一个强健的智能传感器。到第8章结束时，我们将完全掌握智能传感器的设计技术基础并且理解如何基于dsPIC系列DSC来实现它们。

第9章是本书的结尾，将探讨智能传感器和数字信号控制器的未来。还将讨论开放性和专用性协议的优点，以及普遍存在的网络化设备的发展趋势。希望本章能使读者了解到今后设计产品时应当考虑的关键问题。

此外，本书三个附录的内容都是彼此独立的，读者可自行决定是深入研究还是等到需要的时候再读。附录A讲述本书附属资源中的软件，附录B讲解dsPIC器件的初始化以及相关系统的启动代码。附录C将强调系统运行中具有挑战性但又常常被忽略的方面，即中断驱动缓存串行I/O端口，它可用于调试或者普通通信。由于任何一本书都无法包含智能传感器及dsPIC型DSC的所有方面，因此本书附属资源中有一个HTML文件，其中列出了有助于读者进一步学习的其他资源。

最后，有关本书的最新消息以及其他程序实例和资源，请浏览网址www.intelligentsensordesign.com。 18

第2章 直观数字信号处理

到目前为止我们已经看到，智能传感器对终端用户及生产和销售它们的厂商来说多么有价值。现在就要深入了解智能传感器是如何工作的。首先要对DSP理论建立一个牢固、直观的认识。和很多DSP入门教材不同，本书的目标是介绍并使用这些重要概念，而不是推导它们。理由也很简单，就是深入阐述DSP原理需要一本书而不是一章的篇幅。很多作者都已经以一种更加严谨的方式阐述过数字信号处理方面的内容[①]，而本书的目标不是要将这些内容压缩成抽象的要点，而是要掌握如何使用这些重要概念将原始的传感器数据变为有意义的传感器信息。读者在读完本章后，就能轻松地掌握那些在典型应用中需要使用的关键的信号处理技术，并能够确定该采用哪种合理的方法来提取所期望的测量值。

2.1 信号处理的基本概念

尽管这里有关DSP的讨论并不像大多数学术研究那么严谨，但是必须理解关键概念，并以此作为DSP的基础。首先对“信号”和“噪声”给出一个精确的定义，然后在时域和频域分析信号，讲述滤波技术基础。滤波技术主要用于从含有噪声的数据中提取所需要的信息。2.2节将以本节内容为基础并创建一个通用的传感器信号处理架构，我们可以将这个架构应用到各种智能传感的应用当中。

2.1.1 信号和噪声的明确定义

字典[②]中对“信号”一词的解释是电压或电流等波动的电气物理量或脉冲，这种变化代表了某种编码信息，这个定义可以作为理解信号的起始点。电信号有一个有趣的特性就是它们遵循叠加原理。该原理表明，两个或者多个信号在同一时刻通过同一媒介中的同一点的流量，等于各个信号在该时刻分别通过该点的流量之和。例如，如果有N个不同的信号$V_0(t), V_1(t), \cdots, V_{N-1}(t)$，根据叠加原理，信号$Vs(t)$就等于N个信号叠加，可以表示为

$$V_s(t) = V_0(t) + V_1(t) + \cdots + V_{N-1}(t) \tag{2-1}$$

可见，叠加原理是一个十分强大的工具。利用它，我们能够将复杂的传感器信号分解成独立的、基本的成分，从而简化问题分析和最终的系统设计。本书后面介绍的传感器设计实例中，就大量运用了该原理，并针对各自的应用特点设计了合适的信号处理方法。但是，首先还是让我们研究一下叠加原理是如何使所有传感器信号更便于理解的。

考虑图2-1a所示的电路，它在理想条件下将一个热电偶连至电压表。第1章已经提到过，热电偶将产生模拟输出电压$V_T(t)$，该电压与热电偶金属结的温度有关，并随时间t而变化。在本例中，测得的信号$V_M(t)$只代表$V_T(t)$信号的真值，它所附带的编码信息就是热电偶金属

---

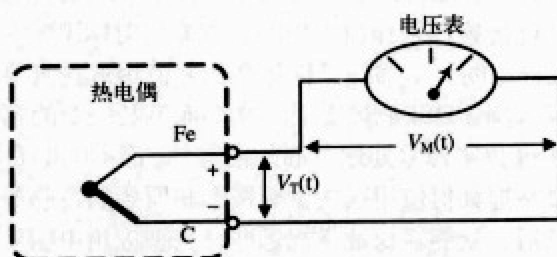
① 介绍两本有关DSP的优秀图书：一本是Steven Smith 所著的《数字信号处理：工程师和研究人员的实用指南》(英文原书书名*Digital Signal Processing: A Practical Guide for Engineers and Scientists*，即将由人民邮电出版社出版)，另一本是Richard Lyons 编写的*Understanding Digital Signal Processing*, 2nd edition。

② *Webster's II New Riverside Dictionary*，1984，Houghton Mifflin Company。

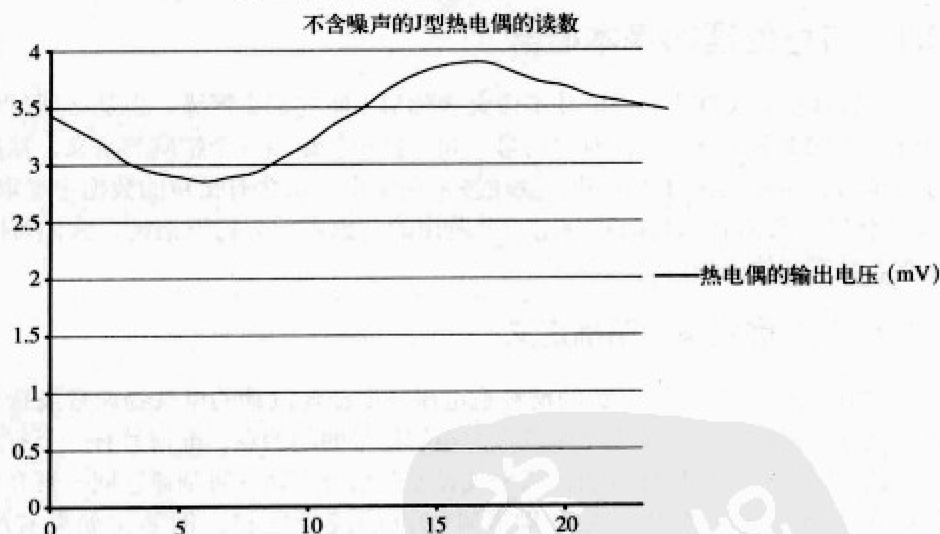
结的温度。

22

但这种理想的环境只存在于我们的想象中，就像完全寂静的图书馆仅存在于图书馆管理员的想象中一样。就像最安静的图书馆也会有听得见的噪声一样，实际的电路中也包含着来自周围环境以及电路所含元件的电子噪声。因此，对基本热电偶电路更精确的表示中还应该包含一个电噪声发生器，它将产生一个噪声电压分量$V_N(t)$，叠加在热电偶信号真值$V_T(t)$上，参见图2-2a。

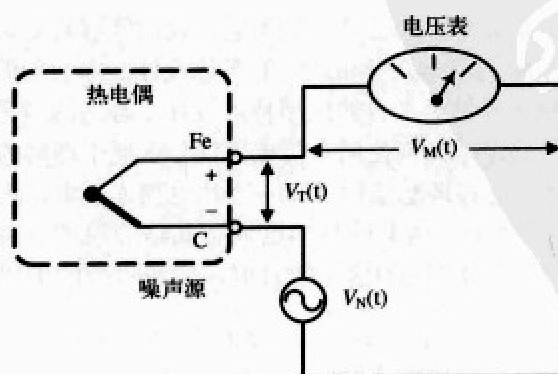


(a) 基本的理想热电偶电路



(b) 理想的热电偶信号的例子

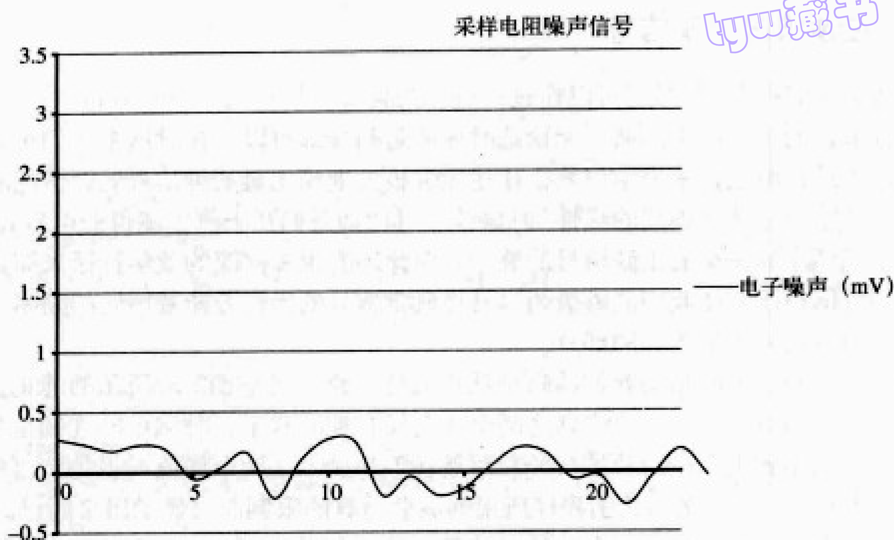
图2-1　热电偶电路和信号



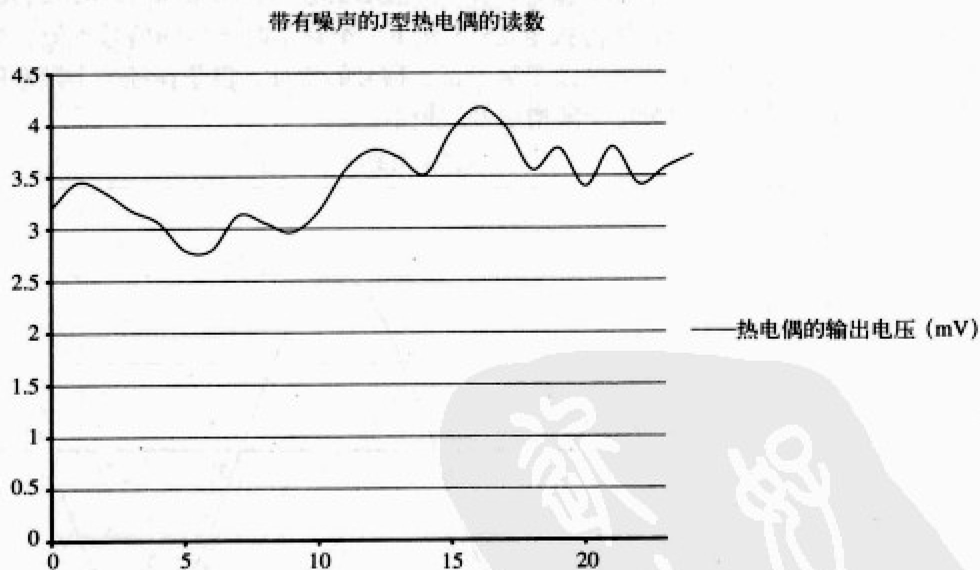
23

(a) 更真实的带有噪声的热电偶电路模型

图　2-2



(b) 电子噪声信号的例子



(c) 带有噪声的热电偶信号的例子

图2-2 （续）

对于一个外置的观测器来说，这种区别并不明显，这种观测器只能看到含有真值和噪声的测量信号$V_M(t)$，它等于

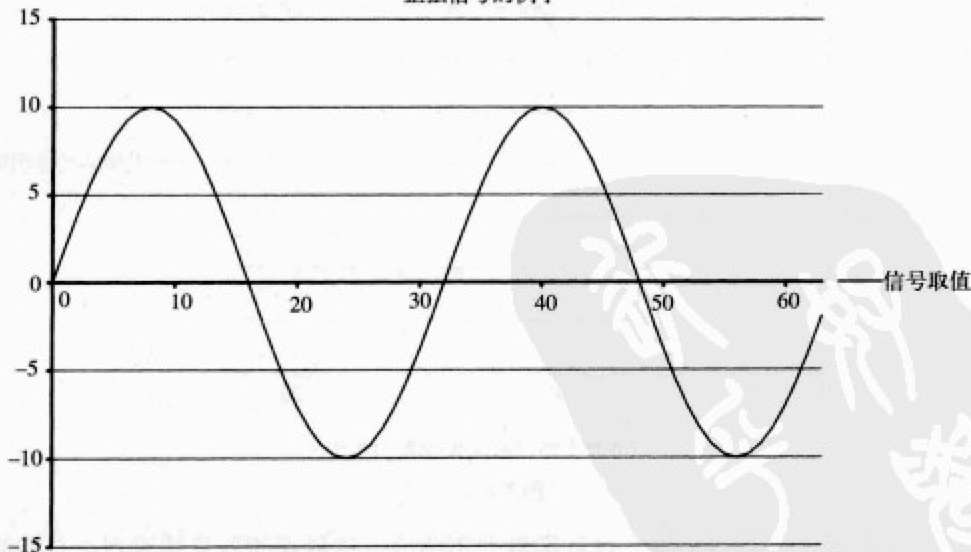
$$V_M(t) = V_T(t) + V_N(t) \tag{2-2}$$

终端用户当然不希望所测得的信号中包含噪声成分，他们需要的是真实的热电偶信号，而不是那些带有失真信息的受污染信号，因为毕竟只有真实的信号才含有他们所关心的信息。根据所关心信号以及噪声的特性，利用下面将要介绍的技术就有可能在含有大量噪声的情况下精确地提取我们所感兴趣的信号。

2.1.2　在频域内观察信号

任何实际的模拟[①]信号都可以通过一种称为傅里叶变换（Fourier transform）的数学运算映射到频域，而选择适当的域（无论是时域或是频域都可以，在时域中我们可以利用示波器或者电压表进行测量）来表示信号，往往能够极大地简化对某种信号处理情况的分析。傅里叶变换的基本前提是，连续的线性[②]时域信号（比如我们在上例中测得的电压）可以精确地表示为不同频率的正交的正弦信号的叠加。对此还有很多高深的数学行话来描述，但是傅里叶变换的价值在于它使我们能够极为容易地确定信号的大部分能量所处的频率，而这实际上是告诉了我们信号中最重要的部分。

通过一个例子就能帮助我们清晰地认识到这一点。考虑图2-3a所示的纯正弦信号，它对应到频域中为图2-3b所示。请注意这两个域之间有趣的联系：时域中的连续信号映射到频域后就变成了频率轴上两个对称的尖峰！尽管图2-3b显示了两个峰值，但是频谱中其实还应当有其他频率成分，只是受到绘图程序所用的离散运算的限制而未显示出来而已。从数学上看，频域中的两个尖峰应恰好位于正弦信号的频率处。如果我们在第一个信号中再叠加第二个正弦信号就可以构建出一个更为复杂的信号，根据叠加原理可知，我们将得到像图2-4a和图2-4b所示的信号。我们在这里看到的就是一个标准的DTMF（dual tone multifrequency，双音复频）信号，它和你在塔奇通按键式电话[③]上按下一个数字时产生的信号类似。增加一个频率的信号后，尽管信号在时域上失去了很多正弦信号的特征，但是在频域上则有四个清晰的尖峰，新增加的两个尖峰就对应于新增信号的频率。 25



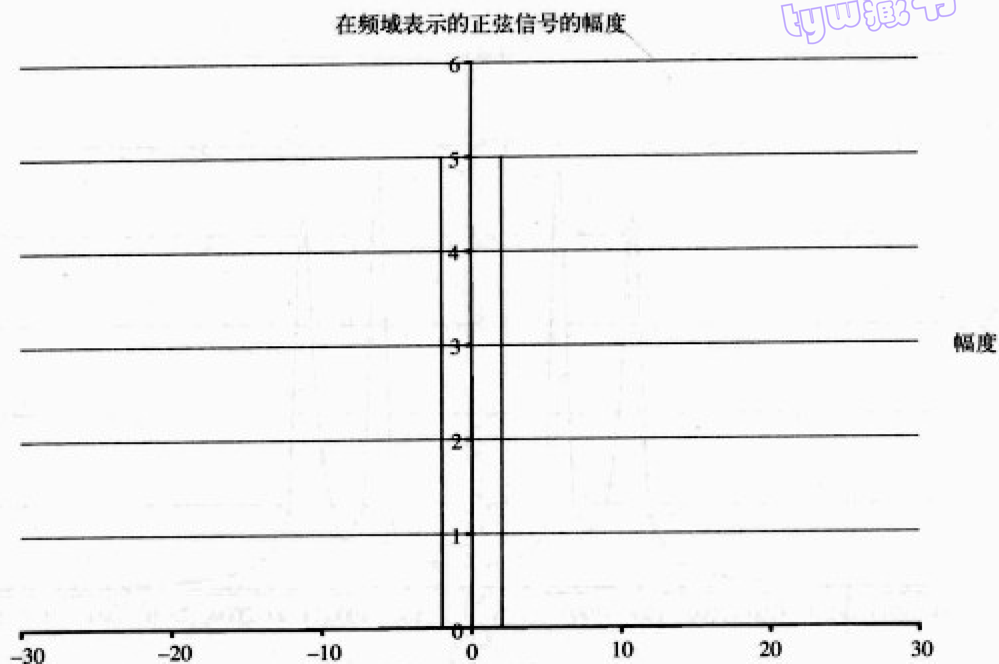
(a)　时域的正弦信号

图　2-3

① 这里所说的模拟信号是时间连续的信号，即未采样的信号。

② 这里的“线性”并非指直线函数，而是指符合叠加原理的函数。从数学上讲，如果将函数变换表示为$\boldsymbol{H}[]$，$y_0(t)$表示该函数对输入$x_0(t)$的响应，而$y_1(t)$表示该函数对输入$x_1(t)$的响应，那么$\boldsymbol{H}[]$是线性的，当且仅当$\boldsymbol{H}[ax_0(t)+bx_1(t)]=a\boldsymbol{H}[x_0(t)]+b\boldsymbol{H}[x_1(t)]=ay_0(t)+by_1(t)$。

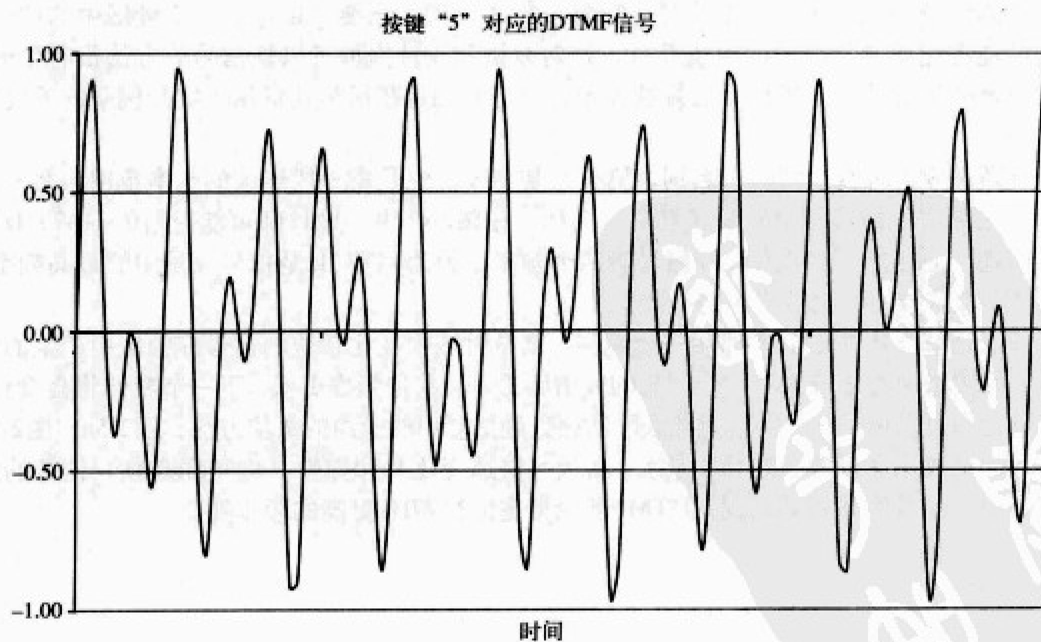
③ 一种每个按键会发出不同声响的电话。——编者注



(b) 同一个信号在频域的表示

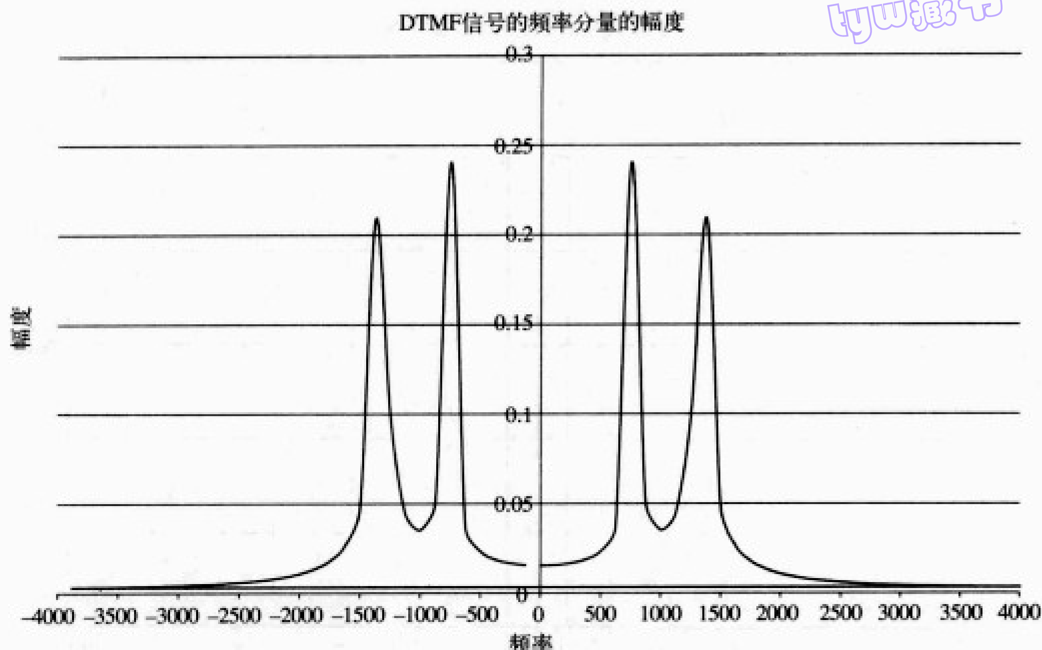
图2-3 （续）

26



(a) DTMF的时域信号

图 2-4



(b) DTMF信号的频域表示

27

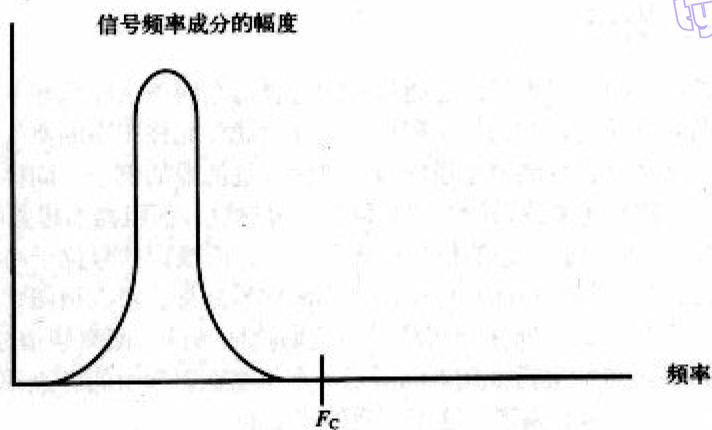
图2-4　（续）

图2-4a和图2-4b显示出傅里叶变换的强大威力，而叠加原理在时域和频域均适用。在时域中相加的信号的频谱等于各个信号分量对应频谱之和。只要简单地看一下频域中的信号，设计师就能迅速确定出信号的组成部分，这对分析和设计提取感兴趣信号的方法很有帮助。所谓的*频谱分析*就是指分析信号的频域表示，这是我们将在后续几章用于实际例子的重要方法。

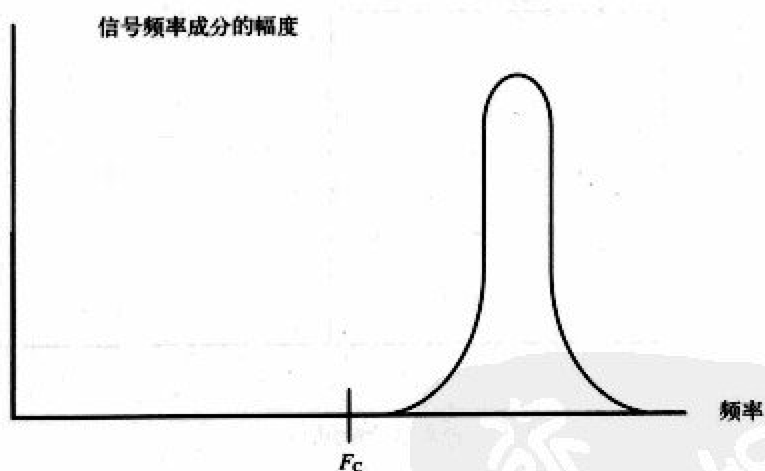
进行频谱分析时常遇到两个名词，第一个是*频带*，它是指一段连续的频率范围；另一个是*带宽*，它通常是指信号中的最高频率。例如，在图2-4b中，设计师可能对770～1477Hz的频带感兴趣，它包括了构成DTMF信号的两个频率。因为1477 Hz是信号分量中的最高频率，所以该DTMF信号的理论带宽为1477Hz。

时域－频域表示中还需要理解一个问题，就是时域中变化较快的信号在频域中产生的频谱更宽，而时域中变化较慢的信号产生的频谱则更窄，并且频率更低。下一节我们将会看到，传感器设计师可以利用这一特点确定去除所感兴趣信号中的噪声的最佳方法。图2-5a、图2-5b和图2-5c分别表示了低频（缓慢变化）、高频（快速变化）和宽带（低频和高频）信号的频域。实际中，信号失真将延续到比DTMF理论带宽值1477Hz更高的频率处。

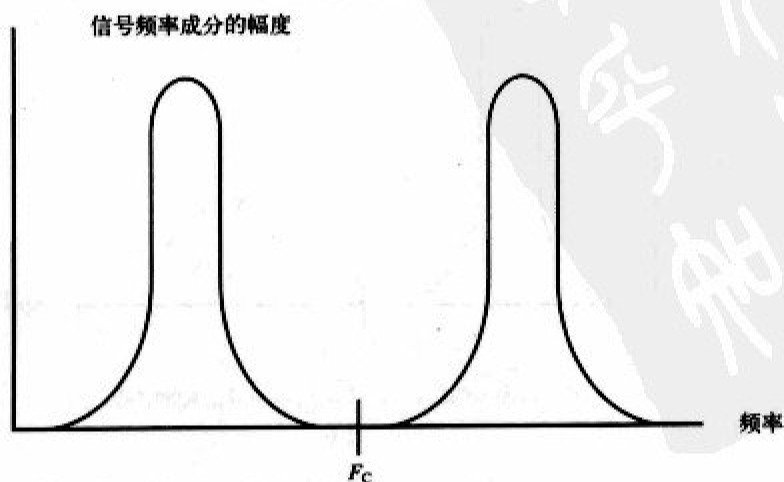
28



(a) 频域中的低频信号



(b) 频域中的高频信号



(c) 频域中的低频和高频信号的组合

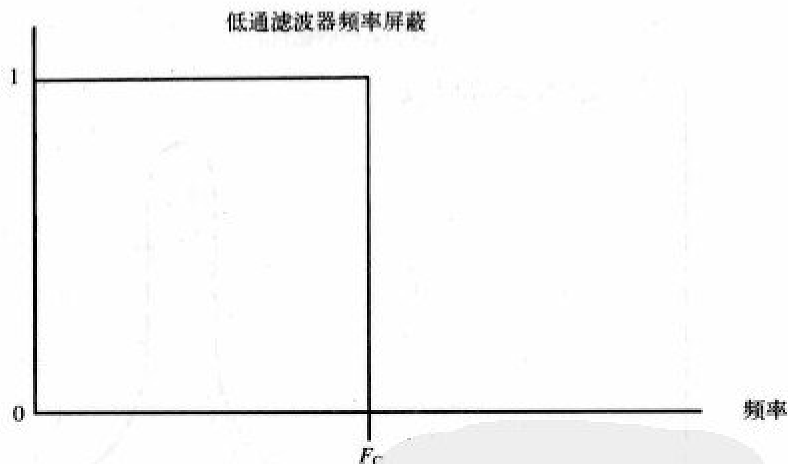
图 2-5

2.1.3 用滤波器净化信号

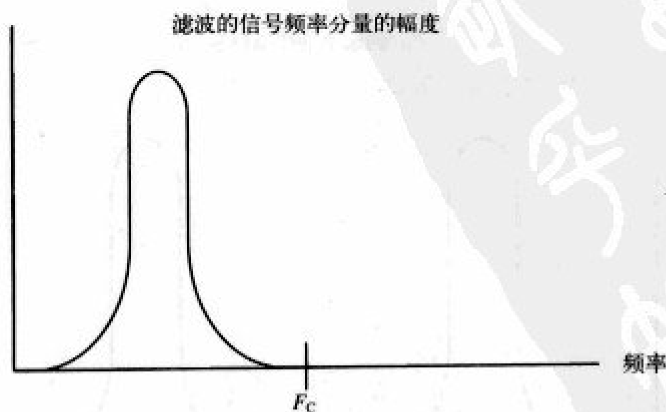
我们都很熟悉过滤器的基本思想：它能从我们想要的东西中去除那些不想要的东西。咖啡过滤器允许液态的咖啡通过，却挡住了碎粒；空气过滤器允许干净的空气通过，却拦截了灰尘和其他污染物。这是日常生活中普通的两个机械式过滤器的例子。同样的概念也可应用于带噪声的电子信号，即只允许感兴趣的"真值"信号通过，而阻挡不想要的噪声信号。

29

从图2-5c可以看出，假设感兴趣的信号位于低频区，而噪声信号位于高频区。那么在理想情况下，我们就能去除高频噪声而只保留感兴趣的信号分量。可以用图来表示上述处理过程，即在频域施加一个屏蔽，从而使低频信号不受影响地通过，而高频信号全部无法穿过。从图中可以看出，这种屏蔽的频谱如图2-6a所示。如果将图2-5c中的各点乘以图2-6a中的对应点，就可以得到图2-6b所示的频谱，这正是我们想要的。



(a) 频率屏蔽的例子



30

(b) 图2-6a所示的屏蔽与图2-5c所示频谱相乘的结果

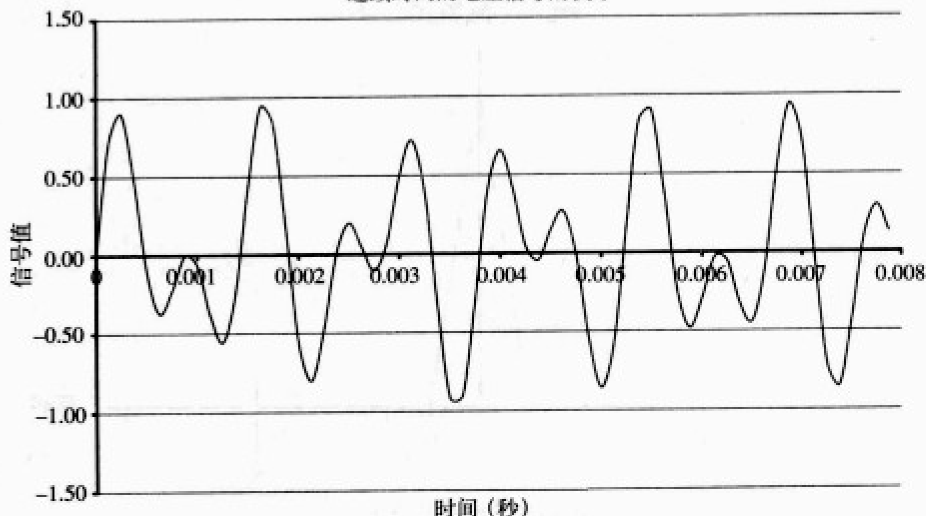
图 2-6

尽管这样的理论实验很有效，但是能否在现实世界中实现上述操作呢？答案是"能"，虽然对实际系统与理想系统之间的偏差有一些重要限制。但在研究这些限制前，先要介绍一个重要的基础概念：采样。

1. 采样模拟信号

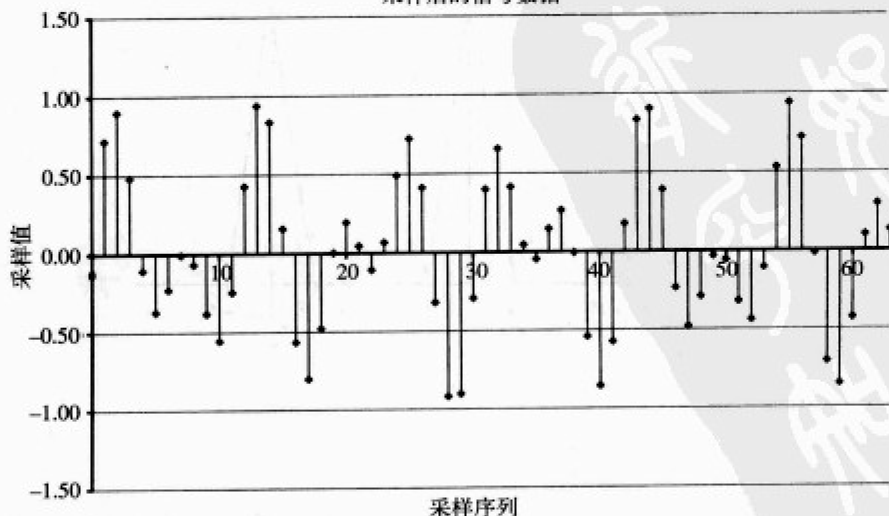
传感器信号本身是模拟信号，也就是说它们在时间上是连续，并且取值也是连续的。但是，直接处理模拟信号需要特殊的电路，这些电路很难设计且造价昂贵，还会因为器件寿命和其他特性变化的影响而随时间发生漂移。另一种更好的方法是，将模拟输入信号转换成数字信号，然后用微处理器处理它们。这就需要进行所谓的**模-数转换**，或称为**采样**。

图2-7a是一个连续时间上的电压信号，而图2-7b是该信号的采样结果。对于那些新接触采样信号的读者，有一个容易感到困惑的关键概念：采样后的信号只是一组数字，且每个数字都对应于某一时刻的连续信号。对于图2-7b所示的采样信号，该信号仅在采样时刻有效。两次采样之间的值不为零，但是习惯上用图表示采样数据时会以直线（或者栅格）表示采样



(a) 连续时间的电压信号的例子

31



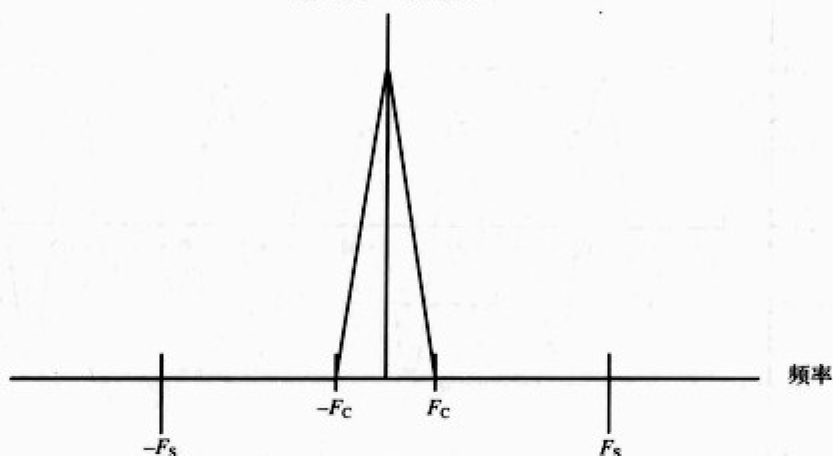
(b) 图2-7a所示信号对应的采样后的形式

图 2-7

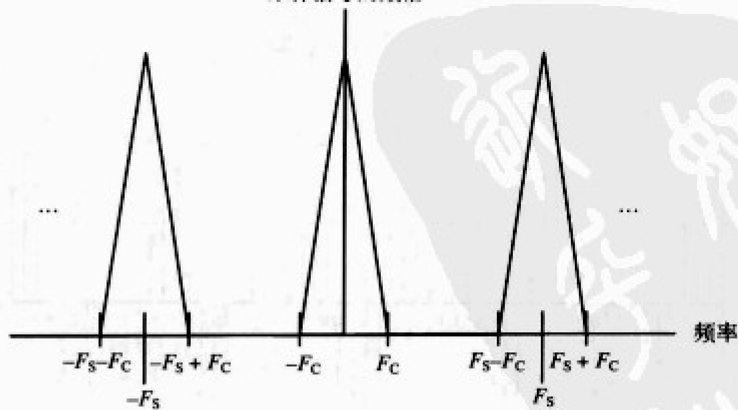
值，其中横轴表示的参数对应于采样时刻（通常是时间或者空间距离）。

另一种形式是用标记号和序列中的采样信号值组合成图。在这种方案中，信号$x(t)$的第一个采样为x_0，第二个采样为x_1，依此类推。如果我们将两个信号x和y相加，那么得出的信号z就等于这两个信号的各次采样值之和：

$$z_0 = x_0 + y_0$$
$$z_1 = x_1 + y_1$$
$$\vdots$$
$$z_N = x_N + y_N$$



(a) 模拟信号的频谱例子



(b) 采样信号对应的频谱

图 2-8

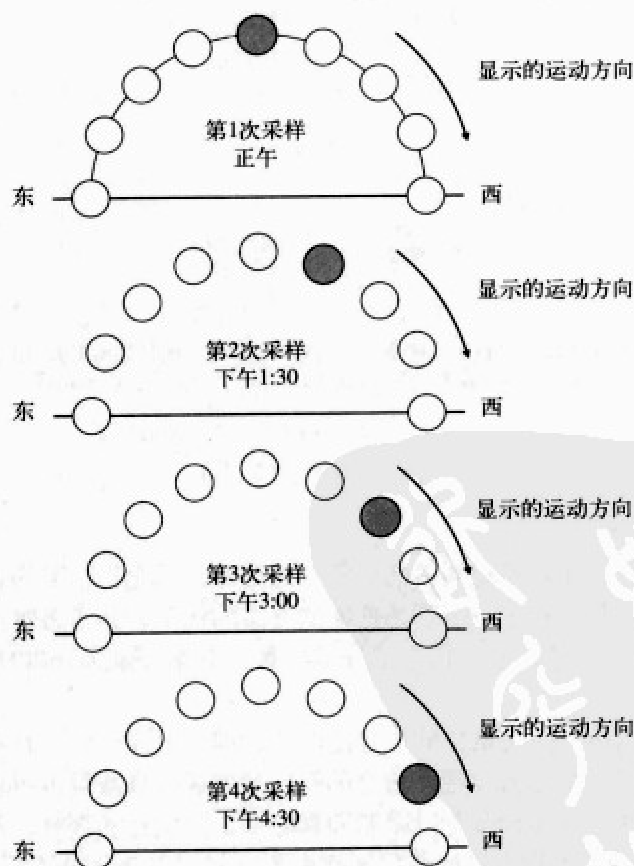
采样对信号有两个重要影响。第一个是所谓的频谱复制，它的意思是采样信号的频谱在频域中会周期性地重复出现，且该周期等于采样频率。图2-8a和图2-8b显示了一个信号本身及其采样值的频谱的例子。

32

显然，当原始信号的最高频率分量大于采样频率的两倍时就会出现问题，这时的采样频率被称为奈奎斯特速率。这种情况下，由于频谱中的部分高频分量与邻近被复制高频谱中的部分低频分量难以区分，被复制的频谱中会有频率分量发生重叠，这种现象也被称为混叠。混叠通常对系统是有害的。尽管现实中无法完全消除混叠，但是还是有一些方法可以将它的影响减小到可以忽略不计。 33

通过一个简单的例子就能明白混叠是如何使我们误以为信号是以某种形式变化，而实际上它是按照完全不同的另一种方式在变化的。设想在一段较长的周期里，采样太阳在一天内不同时间的位置。作为优秀的科学家，我们希望能验证采样速率的确有重要影响，因此决定以两种采样速率获取两组测量值。第一组测量值所采用的采样速率为每1.5小时采样1次，结果见图2-9a。正如我们所期望的那样，测量值显示出太阳在实验过程中由东向西移动。 34

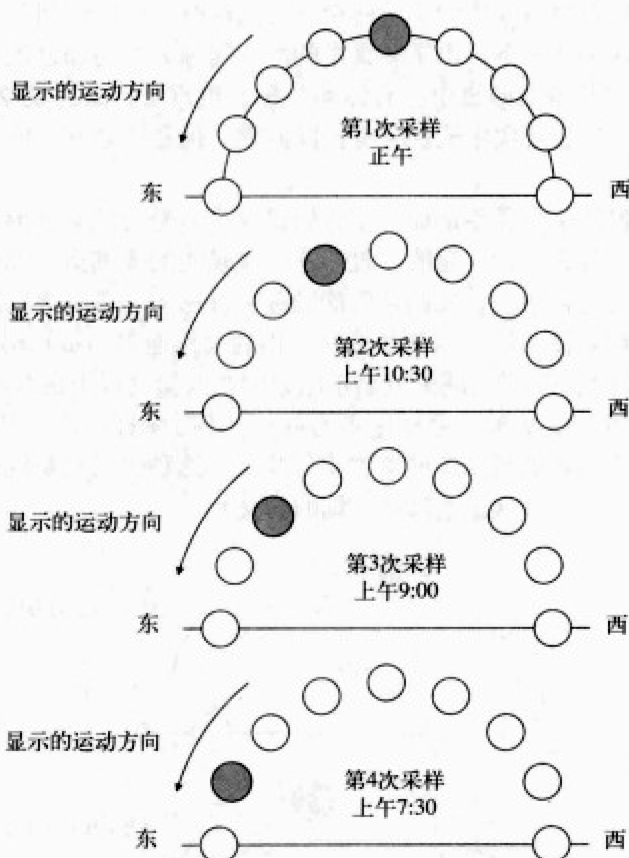
现在我们再看看第二组测量结果，采样速率为每22.5小时采样一次。从数据可以看出，太阳是从西向东移动的，这与我们所知道的事实正好相反！这种错误正是混叠造成的，即信号显现出与实际情况不符的特性（这正是混叠一词的含义）。



当以每1.5小时1次的速率采样太阳的位置时，结果显示出太阳由东向西移动

(a) 每1.5小时采样一次太阳的位置

图 2-9



当以每22.5小时1次的速率采样太阳的位置时，结果显示出太阳由西向东反方向移动。这是因为采样速率不够快，因此我们的采样结果与实际情况不相符

(b) 每22.5小时采样一次太阳的位置

图 2-9 (续)

2. 低通滤波器

我们已经看过一个低通滤波器的例子。这种滤波器允许信号的低频部分通过，同时阻挡信号的高频分量。图2-10是一个理想低通滤波器的例子，其通带宽度（即允许信号通过的频率范围）为1500Hz。请注意，由于滤波器的最高频率分量为1500Hz，因此该滤波器的带宽也是1500Hz。

35

低通滤波器可能是应用最广的一种滤波器，理由很简单：在真实世界中，我们不需要处理无限带宽的信号。低通滤波器对信号的有些频率成分衰减得很厉害，因此消除噪声最常用的方法就是对有用的频率成分加以限制而剔除高出限制点的部分。例如，当我们使用热电偶测量温度时，由于被测物的温度变化速率有限（即温度变化是连续的），因此其输出电压的变化速度也有限。这就意味着温度传感器输出信号的频率分量有上界，超出这个界限的信号都毫无意义。如果我们设计一个能消除所有高于该上界频率分量的低通滤波器，那么该滤波器就一定能消除噪声，因为高于该截止频率的温度信号都无效。

3. 高通滤波器

与低通滤波器互补的是高通滤波器，它只允许信号的高频分量通过，而阻挡低频分量。

图2-11是一个理想的高通滤波器，其通带起始频率为1500Hz，并延伸向无限高的频率。注意，由于该滤波器允许所有高于通带起始频率的信号通过，因此它的带宽是无穷大。

36

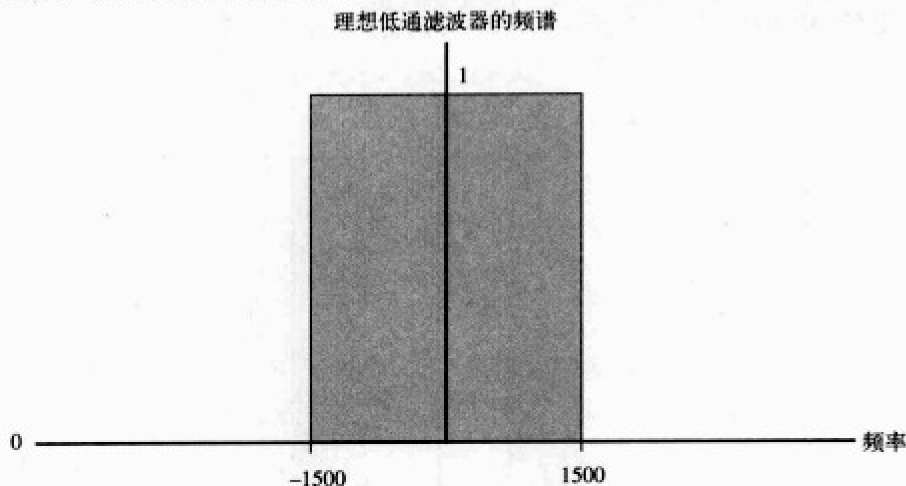


图2-10　带宽为1500Hz的理想低通滤波器

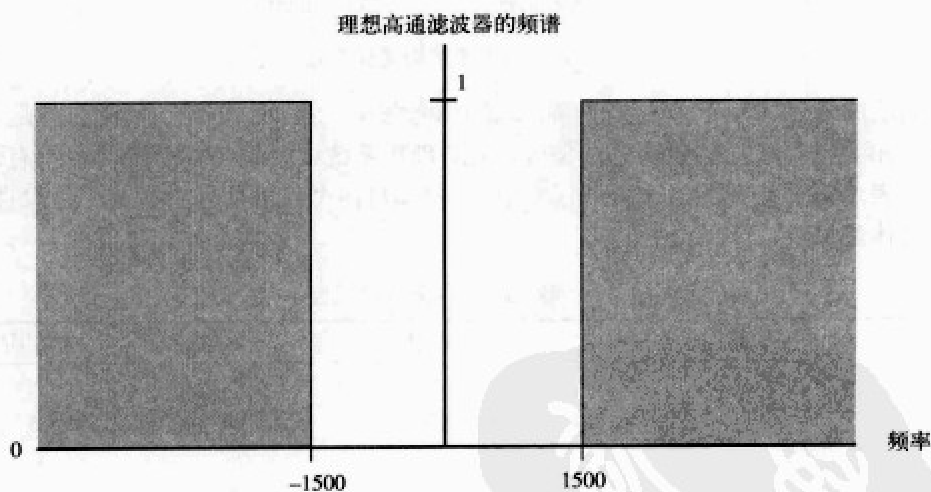


图2-11　通带起始频率为1500Hz的理想高通滤波器

刚刚还提到过，现实中不存在带宽无穷大的信号，那么为何要假设滤波器具备该条件呢？这是因为，有时我们需要测量交流信号，那么根据交流信号的特性可知，任何低于一定频率的成分都是噪声，因为低于那个频率就不存在有效的信号分量。比如人耳的听力就是属于这种情况，人耳只对20Hz～20kHz的频率敏感。任何低于20Hz的声音都没有实际意义，并会被人耳当作噪声。

4. 带通滤波器

带通滤波器实际上是高通滤波器和低通滤波器的组合，其中高通滤波器的通带起始频率比低通滤波器的带宽小，其频谱参见图2-12。这里所示的滤波器的通带频率是从750～1500Hz，滤波器对其他频率都呈阻态。

带通滤波器适用于设计师只希望提取特定频率段信号的场合。最常见的例子就是收音机

的调谐器，它就使用了一个通带很窄的带通滤波器，以便提取某个电台的信号。调谐器的目标就是只允许用户感兴趣的电台信号尽可能清晰地通过，同时又将其他电台的信号（可能是更高或者更低频率的信号）衰减到可以忽略的程度。

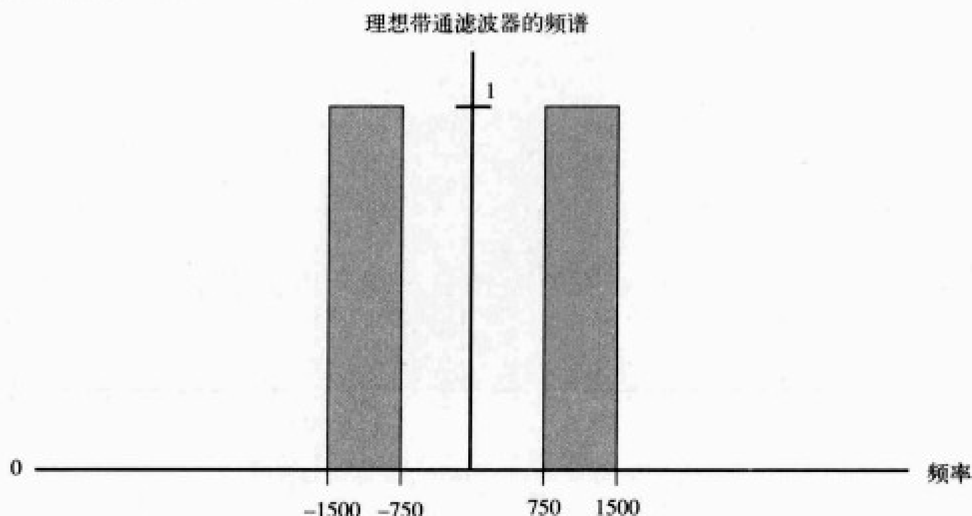


图2-12 理想带通滤波器

带通滤波器还常用于查看某个频带内的信号强度。例如，DTMF检测器就是利用这一原理来确定用户按下了电话机的哪个按键。在DTMF系统中，是用仅有的两个没有公共谐波的频率组合表示某个按键。在这两个频率中，一个取自4个低频信号中，另一个取自4个高频信号中，具体参见表2-1

37

表2-1 DTMF音频组合

	1209Hz	1336Hz	1477Hz	1633Hz
697Hz	1	2	3	A
770Hz	4	5	6	B
852Hz	7	8	9	C
941Hz	*	0	#	D

一般来说，为了构成一个DTMF音频信号，每个频率分量都必须包含大约1.5%的本征值，并且两个信号的强度差不超过3dB。该检测器包括8个分别对应各频率分量的带通滤波器，以及一个允许所有8个信号通过的带通滤波器，它能检测每个滤波器的输出以确定当前时刻哪两个信号有效，这两个有效的信号就构成了组合音（一个来自低频的信号组，另一个来自高频信号组），并且它们的相对强度也满足要求。

38

5. 带阻滤波器

带阻滤波器，也被称为陷波器，它和带通滤波器一样，也可以看作高通滤波器和低通滤波器的组合。带通滤波器只允许窄带信号通过，而带阻滤波器则恰好不允许某个窄带信号通过（即仅衰减某个窄带信号），但是又都不影响其他频率。图2-13就是一个带阻滤波器的例子。

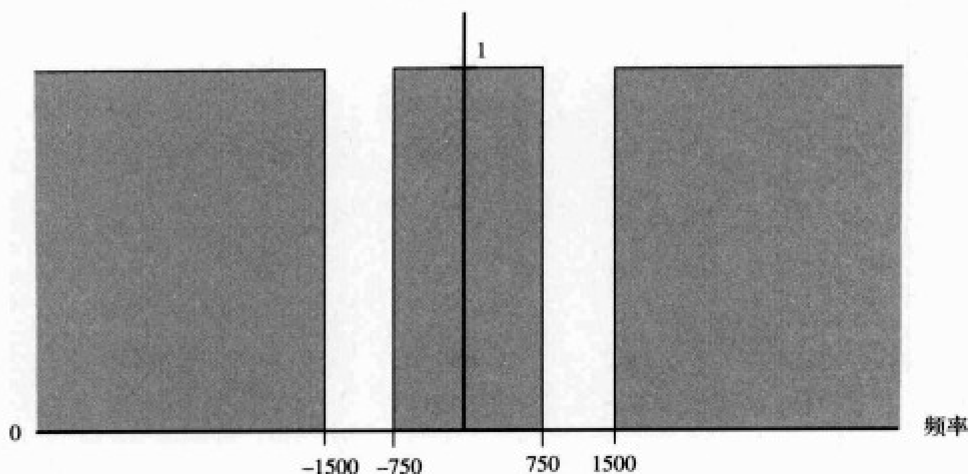


图2-13 理想带阻滤波器

到目前为止，带阻滤波器最主要的应用就是衰减电线上的50Hz或60Hz左右的噪声（根据不同情况有所不同）。在很多应用中，50Hz或60Hz的电力信号会耦合进检测电路，而且糟糕的是，电力信号的频率通常又位于我们所感兴趣的信号频谱的中间。由于简单的低通或高通滤波器会阻止高于或者低于电力信号频率的所有信号，而这可能会同时严重衰减我们所希望获取的信号。因此设计师们就想到使用带阻滤波器，只去除电力信号频率附近的频率分量。

6. 数字滤波器的实现

到目前为止，我们都在严格按照理论线索研究滤波器。下面就研究上述滤波器的数学实现问题。通常，可以利用采样数据加权和的形式来构建数字滤波器。例如，如果已知一组输入信号的采样值为x_i，其中$i=0,1,\cdots,N-1$，那么我们可以设计一个滤波器，其输出y_i为：

$$y_i = a_0x_0 + a_1x_1 + \cdots + a_{N-1}x_{N-1} \qquad (2\text{-}3)$$

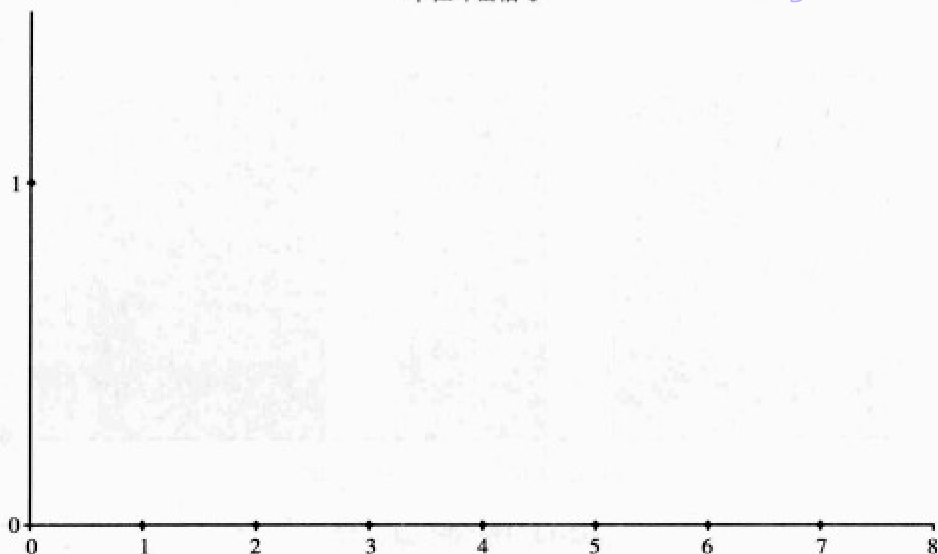
39

其中，a_i项是作用在对应采样值x_i上的常数加权系数。例如低通滤波器就是一个平均器，其输出就等于采样值的平均值。由于噪声会被平均到所有的采样值中，因此它可以实现信号平滑功能。如果我们选择四个采样值以产生滤波后的输出，那么对应的方程为：

$$y_i = 1/4x_0 + 1/4x_1 + 1/4x_2 + 1/4x_3 \qquad (2\text{-}4)$$

通过调节滤波器中各个*抽头*（即采样值）的加权系数就能改变滤波器的响应。为便于设计者使用，很多公司都提供数字滤波器设计与分析软件，并且还在网上提供免费版。本书采用Microchip公司的dsPIC Filter Design™软件来设计和分析我们所需的滤波器。

前面的例子展示的是*有限冲击响应*（FIR）滤波器的结构。采用该方法构建的滤波器的抽头个数固定，因此其输出仅由有限个输入采样值决定。如果让图2-14所示的输入脉冲信号通过一个含有N个抽头的滤波器，那么我们可以知道，由于滤波器的初始输入值是全零，因此它的输出要在开始输入信号后经过N个采样周期才有结果。



40

图2-14　单位冲击信号

另一种结构的滤波器是*无限冲击响应*（IIR）滤波器。它使用加权的输入信号和加权的输出信号共同产生最终的输出信号：

$$y_i = b_0 y_{i-1} + b_1 y_{i-2} + \cdots + b_{i-K} y_{i-K-1} + a_0 x_0 + \cdots + a_{N-1} x_{N-1} \tag{2-5}$$

其中，b_i是施加压在对应的y_{i-1}项上的不变权系数。

初看起来，似乎我们把这个滤波器设计得更复杂了，但实际未必如此。对于前面用FIR滤波器实现的四抽头平均滤波器，也可以用下面的滤波器实现相同的功能：

$$y_i = y_{i-1} + 1/4x_0 - 1/4x_4 \tag{2-6}$$

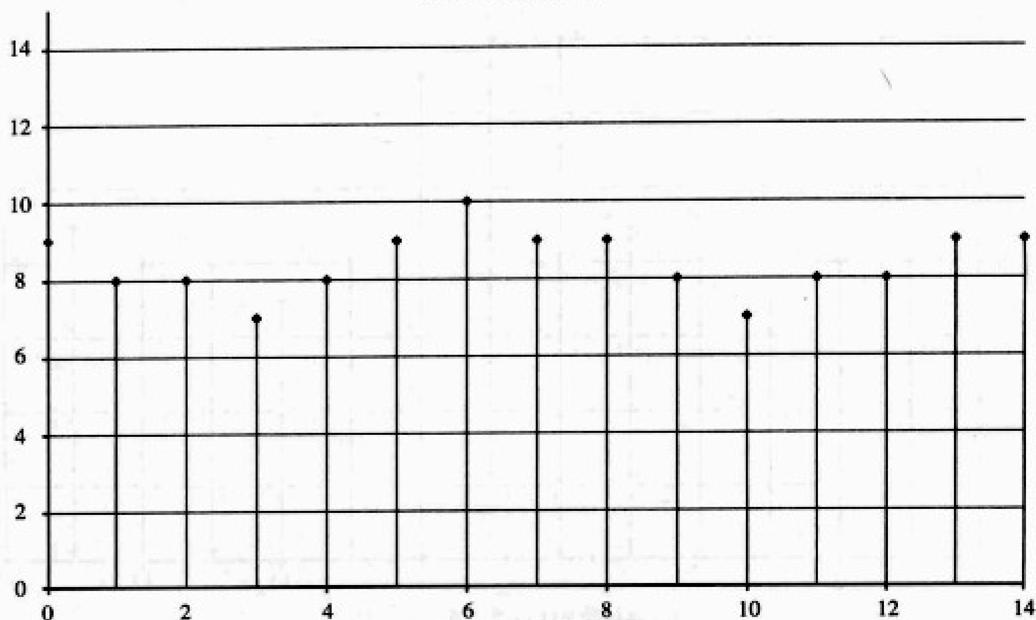
尽管减少一个抽头的计算量可能无关紧要，但是如果利用IIR实现更为复杂的滤波器，就可以看出它能显著降低运算量和内存需求。但是这种简化也有代价，和FIR滤波器不同的是，IIR滤波器在理论上会对输入信号永远做出响应（这正是这种结构的名字的由来），而这些信号可能并不都是我们所需要的。设计师必须仔细核算滤波器产生的累积误差，否则就会使滤波器的性能下降，甚至无法使用。

7. 中值滤波器

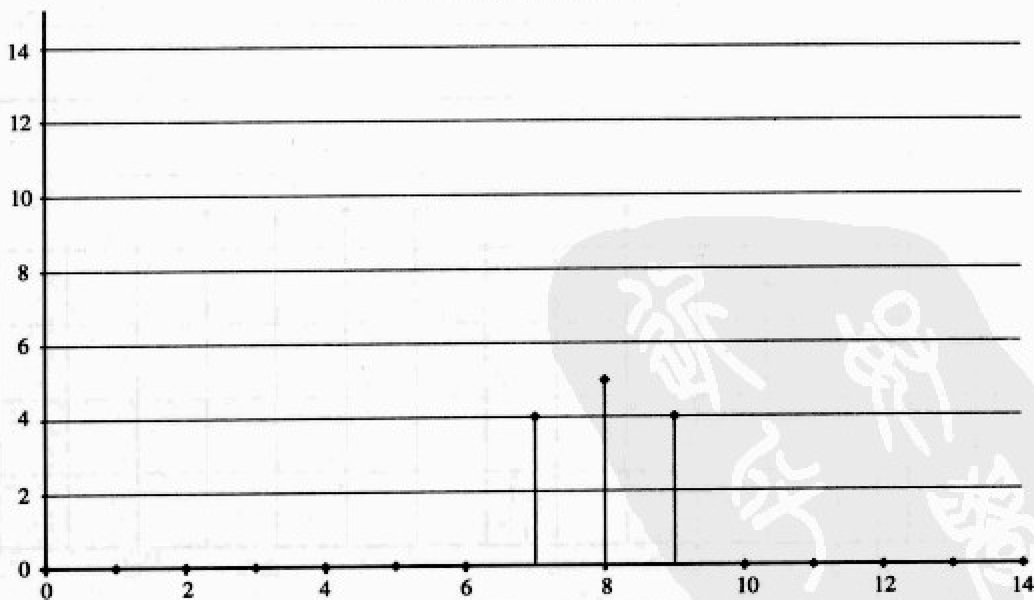
到目前为止，我们讨论的滤波器都是基于简单的数学方程，所以可以运用熟悉且容易理解的方法分析它们的行为。这些滤波器可以出色地工作在带有噪声的特定频谱内，而该频谱正是设计模型。但有时系统对所谓的*散弹噪声*或*猝发噪声*很敏感，这使得所测得的信号带有猝发噪声而不是连续噪声。为了抵消它们，系统必须使用其他类型的滤波，即*中值滤波*，它更具启发性，并能有效地减少散弹噪声。

中值滤波器并不是简单地对被采样信号施以数学运算，而是以另一种滤波器形式对信号取样。首先，将采样值按照由大到小的顺序排序（也可以由小到大排序，关系不大），然后从中选取*中间值*。如果中值滤波器的长度比噪声冲击的长度大，那么噪声信号就会被完全滤除。图2-15a至图2-15d是一个长度为7的中值滤波器实例，它能去除信号中最大长度为三个采样周期的猝发噪声。

41

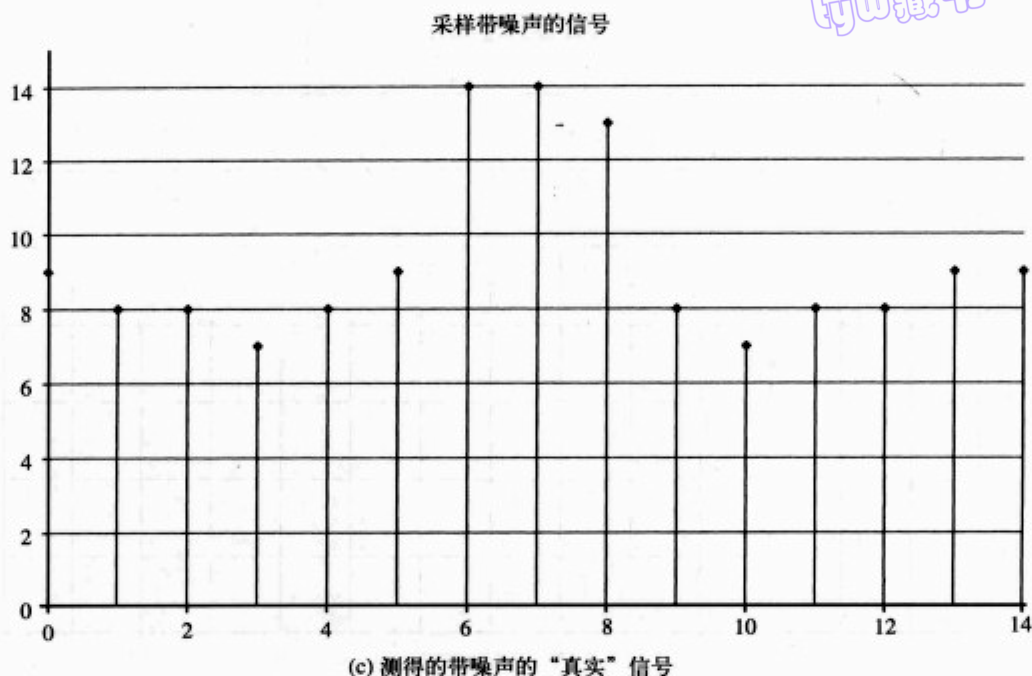


(a) 采样“真实”的信号

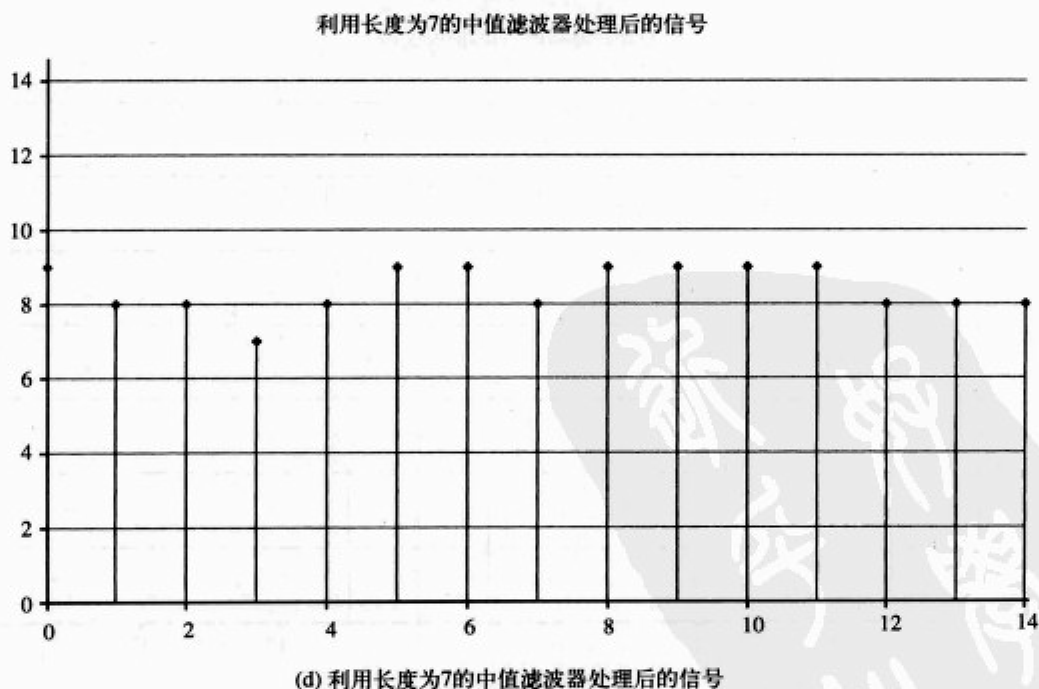


(b) 采样猝发长度为3的散射噪声

图 2-15



(c) 测得的带噪声的"真实"信号



(d) 利用长度为7的中值滤波器处理后的信号

图2-15 （续）

2.2 与信号采样相关的话题

前面已经接触到采样模拟信号时遇到的一个问题，就是所谓的混叠。下面将研究信号采样中还会遇到的其他三个问题：数字化的影响、有限寄存器长度的影响以及过采样。

2.2.1 数字化对采样信号的影响

到目前为止，我们都假设测量的是连续的模拟信号，即测量值是完全准确的。即使是在有噪声的情况下，也有一个隐含的假设，那就是测量值（例如含噪声的传感器输出电压）本身也是精确已知的。在实际中，至少在采用数字信号处理的系统中，上述假设并不成立，因为被测的模拟信号会经过数字化处理，即将模拟信号转换为对应的数字值，处理器再对这些数字值进行数学运算。图2-16展示了该过程（在图示点进行采样）。

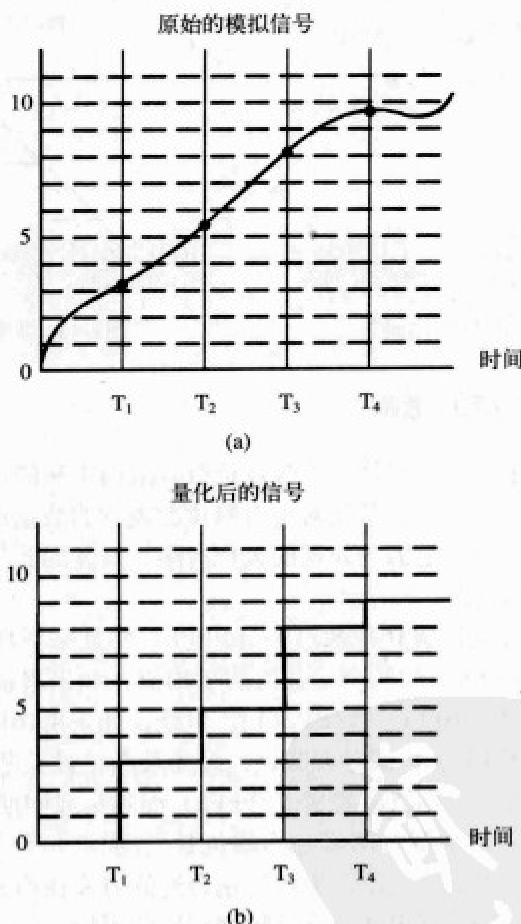


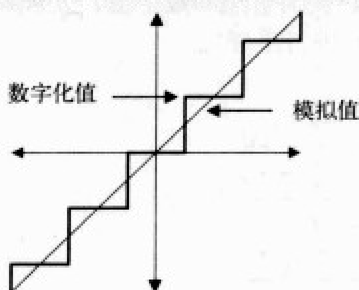
图2-16 四次连续采样的数字化过程

44

数字化带来的问题是，在任何处理器中，只能用有限的位数表示被测信号。例如，假设要采样在0～5V间变化的信号。如果想用1位表示测量值，那么就只能用两个值（0和1）来表示。如果将被测信号的电压记为V_S，那么就可以将信号的低半部分（$0 \leq V_S < 2.5V$）映射为0，而将高半部分（$2.5V \leq V_S < 5V$）映射为1。但这样的分辨率太低了！尽管可以用更多的位数来表示信号以提高分辨率，但是我们永远都只能将某一范围的输入信号映射成特定的输出数字，这就意味着几乎所有落在该范围内的输入信号都有误差（除非被采信号恰好精确对应输出的数字）。

如图2-17和图2-18所示，数字化误差可以看作叠加在测量真值上的噪声。请注意，图2-17是对测量值四舍五入后进行数字化的结果，而图2-18则是对测量值截断后进行数字化的结果，于是

分别得到了三角波噪声和锯齿波噪声。尽管这个问题永远也无法彻底消除，但是我们能够通过采用相对较高的位数（根据应用需要可选用16～32位）来表示算法所使用的数据以减小数字化误差。例如，如果算法使用16位表示数据，就能以0.0015%的精度来表示信号（假设没有其他量化噪声）；如果算法使用32位数据，那么分辨率就要提高到2.3×10^{-8}%（这对应2^{32}个离散等级）。



45

图2-17 近似引入的数字化误差

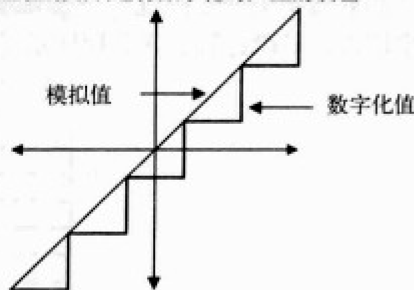


图2-18 截断引入的数字化误差

2.2.2 有限寄存器长度的影响

数字化影响是指由于表示连续信号的数字位数有限而带来的不精确问题。有限寄存器长度的影响与数字化影响联系紧密，是指对由有限位数表示的数据进行重复数学计算所引起的问题。之所以出现这种问题，是因为处理的数位有限，反复的数学运算会导致处理器的累加器溢出。下面举例说明该问题。

假设需要将输入信号量化成16位数（0～FFFFh）并且采用16位累加器。假如需要计算位于量化范围3/4处（BFFFh）的两个采样值的平均值，如果累加器的精度足够，那么计算结果就是BFFFh[1/2(BFFFh+BFFFh) = BFFFh]。但是，如果用16位的累加器计算这两个采样值之和，那么结果不是17FFEh，而是7FFEh，因为最高位被截断了（累加器的空间不足）。如果再计算和的一半，那么平均值就变成了3FFFh，而不是我们期望的BFFFh。

尽管可以使用更大的累加器（即累加器的位数更高），但是该问题是系统的固有问题，并且在进行乘法运算时显得更为突出。通过采用合适的方式在内部表示数据，并且小心地处理上溢出和下溢出问题，设计师也能消除有限寄存器的影响，但是该问题必须反复强调，以避免系统出现灾难性故障。第3章将介绍dsPIC处理器是如何解决该问题的。

2.2.3 过采样

前面已经提到过，设计师必须确保采用DSP的系统对输入信号的采样速率大于奈奎斯特率（它等于输入信号最高频率的两倍），以防止出现混叠。实际中，由于实际的A/D性能与理想A/D性能存在差别，采样速率至少应是输入信号最高频率的4～5倍。这样会进一步加大采样信号频谱各部分的间隔，使彼此相碰的可能性最小。

46

还有一个所有滤波器都存在的问题：滤波器输出信号与其输入信号之间的时延问题。这也是数字滤波器和模拟滤波器都存在的问题。通常来说，滤波效果越好，时延也越大。如果时延太大（这与应用的场合有关），那么滤波后的输出就可能没有用处，因为它到达时间太

晚了，系统的其他部分已经无法使用它。

过采样是指实际的采样速率可能比最低要求速率高得多，从而允许信号成功滤波并且产生的时延也不大。过采样信号的滤波效果好，由于此时的时延相对较小，并且实际采样速率比所需采样速率高得多，因此滤波后的信号也可以被系统的其他部分实时地采用。该方法的缺点是：过采样需要更高的处理功率，以应付更高的数据处理速率，这通常会增加系统成本以及功耗。

2.3 如何分析传感器信号的应用

在分析具体传感器信号处理应用时，设计师需要了解系统的以下方面：

(1) 被测信号的物理特性；
(2) 被测信号的物理特性与对应传感器输出量之间的关系；
(3) 感兴趣的信号的频谱以及环境噪声源信号的频谱；
(4) 运行环境的物理特性；
(5) 可能导致误差的因素以及应对方法；
(6) 标定要求；
(7) 用户或系统接口要求；
(8) 维护要求。

通常，掌握被测物理参数和工作环境的特性有助于设计师确定传感器系统所需的合适的信号处理方法。例如，如果测量一个用小加热部件加热的大块金属的温度，那么最好假设感兴趣的信号的频率成分很小，因为被测物的温度只能缓慢变化。这意味着传感器可以对输入信号进行强滤波以去除噪声。相反，如果测量一个被激光加热的小型部件的温度，那么传感器必须能够对温度的瞬时变化做出快速反应。在这种情况下，必须弱化噪声滤波，可能还需要通过其他处理方法去除通过初始滤波器的噪声。

47

了解被测物理量与传感器报告给用户和系统其他部件的对应参数之间的关系也很重要。传感器报告的参数值与待测物理量之间是否呈线性关系（比如RTD型温度传感器）？或者存在非线性关系（比如热电偶）？如果是非线性关系，那么是否可以将它变成分段线性的，以简化运算？如果对物理量和报告参数之间的关系理解不透彻或者错误，就可能导致设计的传感器系统毫无用处。

最后，设计师必须始终牢记传感器系统是应用于实际环境中的，一定会遇到很多问题。所设计的系统应当能够检测普通故障。错误检测能力越强、处理故障的能力越强，系统性能就越好。传感器如果在产品生产之初就出现故障，那么很可能导致整个生产线停产，因此任何能够快速排除故障并且易于维修的设计都会得到最终用户的极大赞赏。但是，比维护更重要的是传感器系统检测可能导致系统无法安全工作的危险情况的能力，并且在危险排除后又能继续工作。在设计能够工作在故障导向安全环境中的传感器时必须高度重视故障检测、报告及维修问题。

2.4 一个通用的传感器信号处理架构

下面将建立一个通用的传感器信号处理架构，该架构将用于本书第4章至第7章的应用设计实例。和很多优秀设计一样，该架构貌似简单，但是其关键之处在于如何可靠地实现它，并且始终都能精确、按时地完成指定的任务。图2-19是该架构的流程图。

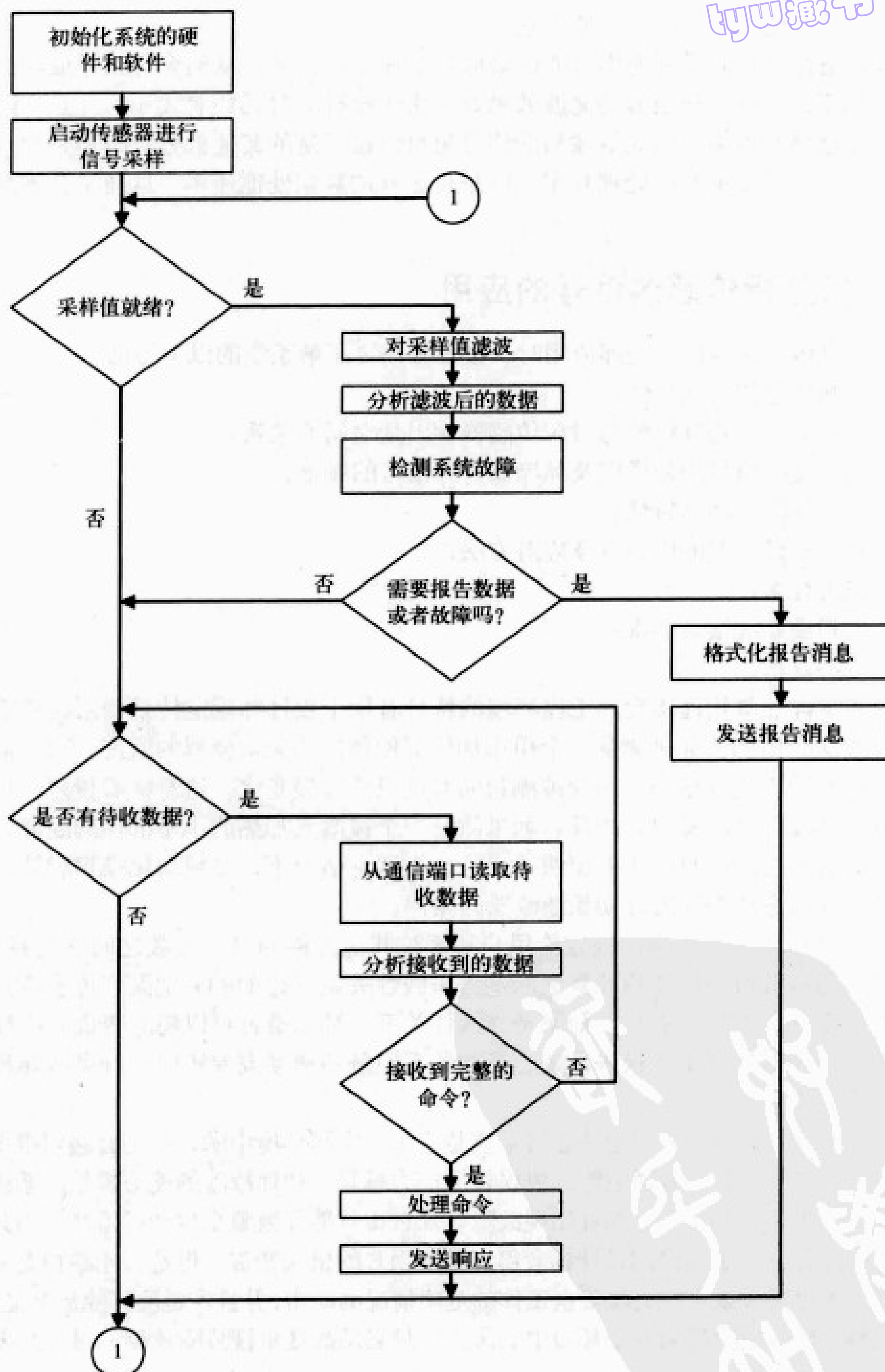


图2-19 通用的传感器信号处理架构

该架构必须设计成硬实时系统，即它对系统输入和输出的响应时间必须是确定的（在固定时间内响应），并且必须在下次输入或者其他事件发生前完成对本次输入信号的所有处理，至少应完成核心部分的处理。而非核心部分，比如通信协议处理部分也很重要，但是它们是

软实时处理的。软实时系统最终也必须能够处理所有输入和事件，但是允许它将这些输入事件排列成队，以便系统在合适的时候继续处理。

48~49

2.4.1 信号调理与获取

信号调理与获取部分负责对模拟输入信号进行必要的调理，将信号的频谱限制在便于处理的频带内，并将信号放大到便于数字化的等级并将最后得到的模拟信号数字化。这部分输出的是一组采样数据流，它们将被系统的后续部分实施数字式处理。

2.4.2 预分析滤波

一旦采样到原始物理量的信号，那么通常必须结合应用的特点对其进行滤波，以去除不需要的噪声或者将信号整型成更加有用的形式。这种滤波通常在获取采样值之后立即进行，以便信号链中后面的处理算法能够使用到相对干净的数据，从而可能得出更好的结果。

2.4.3 信号线性化

有时，我们感兴趣的参数并不随被测物理量呈线性变化。一个常见的例子就是热电偶信号，它的输出电压与对应的温度之间存在复杂的多项式关系。在这种情况下，必须对信号进行线性化处理，以便于后面的参数分析部分使用。根据被测物理量的特点不同，线性化技术也有所不同。

2.4.4 参数分析

参数分析算法与应用的关联性也很强。尽管参数分析在理论上仅受设计师的想象力的限制，但是有一些是典型的处理，比如参数变换（在数学上将被测信号转换成期望的与参数值对应的形式）、频率分析以及阈值比较等。通常，这是整个传感器系统中最为复杂的部分，也是最能提高产品价值的部分。

2.4.5 后分析滤波

一旦计算出了参数值，通常需要对计算结果进行平滑滤波，然后再将它们用于系统的其他部分。和预分析滤波一样，后分析滤波也是针对应用而设计的。

2.4.6 故障检测与处理

50

参数分析部分是传感器系统中最有价值的部分，而故障检测与处理部分则是关乎系统能否生存的重要部分。根据产品检测故障和排除故障的能力，特别是在故障后果极为严重的场合下的应对能力，可以区分出它们的优劣。简单的故障检测可能包括检测传感器元件是否在位，以及验证所提取的参数值是否在合理的范围内等。更高级的故障检测则可能在实际故障发生之前就能诊断出来并且向用户报警。

2.4.7 通信

该架构的最后一部分是通信。这部分用于报告智能传感器产生的所有信息并且可以用于配置智能传感器的工作模式，因此其接口的稳健性和可靠性极为关键。尽管并非所有的系统

都支持所有的通信接口，但是可使用的通信接口还是有很多种，包括RS-232、CAN、以太网以及无线等。设计师必须在能够满足产品的成本和可靠性要求的前提下，选择最便于产品与系统其他部分集成的通信接口。

2.5 小结

本章我们介绍了数字信号处理的基本概念。读者可能已经认识到，为了研发出稳健的传感器系统，其实并不需要透彻掌握DSP方面的知识，只需对其有一个直观的（不必是彻底的）理解即可。尽管只掌握了以上知识，我们就已经能够开发出适用于数字式分析和报告传感器信息的通用架构。后续几章将把此架构用于设计适用于特定领域的传感器系统。

51

第3章 dsPIC系列DSC揭密

通常，人们旅行时所选择的交通工具主要取决于旅行目的及其实现方式。比如，尽管理论上我们可以用跑车将数千台电脑运送到某个配送中心，但是使用拖车来完成这项任务显然效率更高。实现智能传感器也与此类似。如果我们所使用的硬件平台是针对所需完成的任务而专门设计的，那么就更容易达到要求的性能指标。通常，这种平台必须具备确定[①]的监控信号采集、滤波和分析功能，同时又能可靠地处理与外界的所有通信工作。在很多应用中，系统都需要处理多路信号和多通道通信任务，这又提高了对系统处理性能的要求。

近来出现了一种名叫*数字信号控制器*（Digital Signal Controller，DSC）的新型处理器，它集成了纯数字信号处理器的强大的数学运算功能以及标准微控制器的判断功能。Microchip公司就出品了dsPIC®系列DSC，它在单枚芯片上集成了很多功能，从而使设计师能够在非常小尺寸的情况下设计出强大的测控系统。接下来的几章，我们将在各种常见的智能传感器设计中使用dsPIC系列DSC。但在此之前，我们必须了解芯片为用户提供了哪些资源。研究该问题的目标有两个：一个是要掌握芯片具备哪些功能；另一个是要理解dsPIC的设计师为何要设计这些功能。这样我们就能利用芯片提供的这些资源得到最佳的系统设计。

研究dsPIC系列DSC的方法可能很多，但是其中最有用的是三方面：理解芯片的数据处理结构、研究芯片支持的数学表示和运算以及分析片上各种外围设备资源。全面理解这三个方面将为基于该芯片设计出实用的系统打下坚实的基础。

dsPIC系列DSC实际上是具有不同配置（用户可以根据需要选择最适合自己应用的外围设备组合）的两个相关芯片系列，因此，直接称之为“一个dsPIC芯片”是不合适的。为了更贴近实际应用，本书只详细介绍其中的一款芯片——dsPIC30F6014A，它几乎集成了该系列芯片具备的所有片上外围设备。我们研究的只是该芯片的主要方面，有关它的文字资料可能有数千页，这里涉及的只是其中的很小一部分。这里讲述的内容都可以在Microchip公司提供的标准文档中找到。但是，对于那些并不熟悉Microchip产品的人可能并不容易找到这些内容（有时甚至我们自己也难以找到）。本章的价值并不在于它介绍了新信息，而在于它以一种易于学习的方式将相关信息进行了重新组合。具备了这些知识后，我们就能正确地使用这种芯片，并且具有扎实的技术基础，从而能够在需要的时候更深入地研究芯片的功能。

在下面的讨论中，我们会偶尔遇到dsPIC系列DSC芯片的实际功能与芯片资料上讲述的不一致的情况。这种差别（就是所谓的*勘误表*）常会使设计师怨声载道，因为这意味着芯片的工作方式与资料上的叙述不符。从设计师的角度看，主要有两种错误：一种是周边型错误（一种技术术语），另一种是非周边型错误。周边型错误（通过编程或者硬件设计可以克服）会使芯片使用不便，但是还不会导致致命的问题，而非周边型错误，特别是对于需要这些功能的场合，就可能严重延误项目开发进度并增加开发成本，甚至会导致失败。

---

① *确定性*系统的行为要求是指设计必须保证系统能够在规定的时间内对事件做出响应，进一步的隐含约束还包括响应时间必须适合于系统将要完成的具体任务。比如每3ms就会发生一次事件，如果系统的响应时间长达3s，那么这显然是不符合要求的。

因此，明智的设计师总会查阅所用芯片或该系列芯片的勘误表。各家半导体公司发布勘误信息的方式也有所不同，Microchip公司通常会在勘误说明书中完整地说明所有已知的问题，并将它作为对应芯片的补充数据手册或编程参考指南。这些勘误都和对应的数据手册一同发布在公司网站（www.microchip.com）上。在开始设计之前和调试阶段查阅这些勘误信息，可以使设计师避免自己受挫、使公司项目不延期并降低客户的成本。尽管可以怀疑开发者设计的代码或者硬件有错误（至少在开始是这么怀疑的），但是当我们辛苦地解决某个问题并且似乎只是芯片工作不正常时，那么最好还是查阅勘误表或者咨询制造商。因为你的判断可能是对的！

54

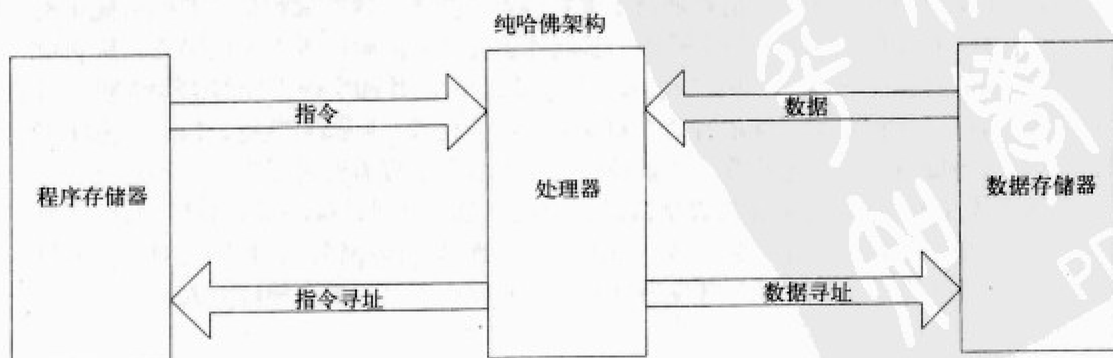
3.1 dsPIC系列DSC的数据处理架构

dsPIC系列产品是针对高速、强调数学运算的处理应用而专门设计的，它具有专用的硬件单元，能够以最小的CPU占用率完成高速处理。但是，就像机械发动机的各个部件必须按预定方式协同工作才能产生动力一样，dsPIC DSC各个单元的运算也必须协调，否则芯片就无法到达期望的处理能力。为了生成必要的协调性，设计师必须透彻理解芯片的架构以及片上集成的各个单元。

3.1.1 dsPIC系列DSC的存储器

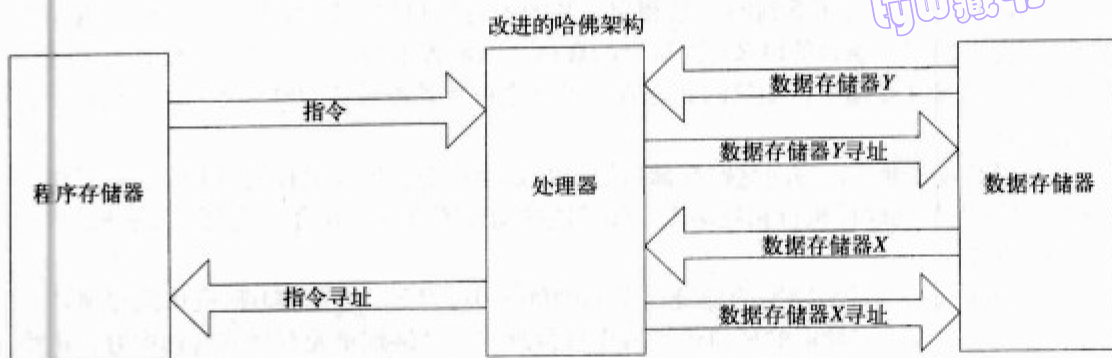
要掌握dsPIC系列DSC，首先就要了解它的存储器架构，因为这是影响数字信号处理系统的数据吞吐量指标的关键因素之一。dsPIC系列DSC采用改进的哈佛架构，具有独立的程序存储器总线和数据存储器总线，从而使处理器能同时提取指令和指令运行所需的数据。

在纯哈佛架构中，这两种总线是完全独立的，彼此间无法交换数据。尽管哈佛架构是完全逻辑化的方法，但是由于它在每个指令周期内只能提取一个数据，因此还不能满足很多数字信号处理应用的需要。通常情况下，需要在每个指令周期内提取两个数据，一个是采样值，另一个是与采样值相乘的系数，这样，在纯哈佛架构中就需要两个指令周期才能完成。为了克服上述限制，改进的哈佛架构实际上有三套总线：一套用于提取指令（即程序存储器总线），另两套则用于提取相关数据（在代码注释中被表示为X存储器总线和Y存储器总线）。这样就可以实现真正的单周期操作，在相同时钟速度下，改进的哈佛架构的效率是纯哈佛架构的两倍。图3-1a和图3-1b显示了纯哈佛存储器架构和改进的哈佛存储器架构的差别。



(a) 纯哈佛架构

图 3-1



(b) 改进的哈佛架构

图3-1 （续）

我们在本章的引言中已经提到，数字信号控制器集成了纯DSP的数学运算能力和微控制器的决策能力，而这种能力反应在dsPIC系列DSC身上，就是它在进行高强度的数学运算（比如乘积－累加类指令，MAC）和执行其他类型指令（有时是指微控制器类指令）时，使用了不同的存储器模型。这样做使得dsPIC系列DSC能够采用最适合某段指令的存储器结构；这种双数据总线架构能够显著提升那些进行数学运算时需要使用多个数据源的应用系统的性能，而对于没有这种需要的指令则可以统一地看作单数据空间。3.1节最后会讨论有关寻址方式和地址发生单元的内容，会详细描述如何使用两种架构来访问存储器。 [55]

但是，dsPIC系列DSC还缺少一种在纯DSP和高性能微控制器中常见的架构：多级指令流水线。指令流水线可以将处理器即将执行的命令在内部排序，进而对其预解码以加速执行过程。尽管这种方法适合于那些数据处理过程就像固定的水流一样且不会被中断的应用（并且已经取得了成功），但是对于那些需要经常改变指令执行顺序的系统（特别是中断驱动的系统），流水线架构则会严重影响处理效率。当处理顺序变化很大时（例如，当处理器需要跳转到流水线以外的某个地方时），就必须清空流水线中的已有内容，直到载入新的、正确的指令后才能继续执行。我们可以用一个交通方面的例子来说明这个问题。这就像在高速公路上驾车一样，如果一直以最高限速在高速公路上行驶，那么用不了多少时间就能跑很远。但是，如果驾驶员频繁驶出再返回高速公路，那么行驶相同距离所花费的时间就长得多。 [56]

尽管dsPIC系列DSC没有多级指令流水线，但是它可以用单级指令预提取机制：在执行指令之前的一个指令周期完成对下一个指令的读取和部分译码功能。这使得大多数指令都能在单周期内执行完毕，从而显著加快系统的决策速度，因为如果不需要执行预期的指令，那么重新载入新指令的速度要快得多。

1. 数据空间存储器映射

dsPIC器件的数据存储器可分为三大类：特殊功能寄存器、静态RAM（SRAM）和映射到数据存储器空间的程序存储器。

a. 特殊功能寄存器

dsPIC系列DSC利用一些存储器映射寄存器来配置、控制和检查器件的各种运行状况。这些寄存器被称为特殊功能寄存器（SFR），它们占数据空间存储器映射的低2KB。和通常用于存储数据的标准数据空间存储器不同，SFR内的数据是位映射的，因此我们能对寄存器的某一位进行读写操作，以命令DSC完成特定动作。

例如，我们将在后面看到的，当利用芯片的串行端口传输1B数据时，需要应用程序先按照一定顺序向串行端口的SFR写入适当的数据，以配置串行端口的工作参数。类似地，应用程序还可以通过查询串行端口的状态寄存器（它也是外围设备的SFR之一）来检查是否接收到新数据。

处理SFR的关键是，访问这些存储器地址时必须小心，因为对任何SFR的读写操作都可能导致dsPIC芯片的硬件执行相应动作，如果这些动作不合适，就会导致意外的后果。

b. SRAM

dsPIC30F6014A的SRAM的地址位于0x0800～0x27FF，共有8KB随机访问存储器。和SFR不同，对这部分存储器的任何读写操作只会改变某个地址单元存储的数据内容，但绝不会影响dsPIC芯片的运行。应用程序可以利用SRAM存储那些在软件运行过程中会发生变化的数据，比如滤波变量或者是用于通信的接收缓冲区。

57

在特殊条件（后面将会提到）下，还可以将SRAM分成两个独立的数据段（数据段*X*和数据段*Y*），以提高数据吞吐率。这两个数据段的起止地址是由dsPIC器件的硬件固定的。对于dsPIC30F6014A来说，其数据段*X*的地址范围是0x0800～0x17FF，而数据段Y的地址范围是0x1800～0x27FF。

c. 映射到数据空间存储器的程序存储器

在有些算法中，特别是数字信号处理算法中，常常需要对变量乘以固定的系数。由于这些系数是固定不变的，因此，如果可以将固定系数存储在程序存储器内并且数据处理硬件还能调用固定系数，那么将节省宝贵的SRAM空间。dsPIC的存储器架构能够方便地实现上述功能，我们将在程序存储器空间一节详细介绍。

2. 程序空间存储器映射

dsPIC系列中各器件的资源配置不同，因此它们的存储器配置也不同。设计师应根据器件的数据手册来查看具体的存储器映射范围。下面将以dsPIC30F6014A为例介绍存储器映射，dsPIC系列中应用的程序空间存储器映射都会在特定内存地址方面存在细微差别。

如前所述，dsPIC系列DSC含有程序地址空间和数据地址空间，而数据空间又可分为数据空间*X*和数据空间*Y*。程序空间存储器包括最多4MB的24位指令字，但是其中有些地址空间不允许用户访问。程序存储器本身又分为用户存储器空间（000000H～7FFFFEH）和配置存储器空间（800000H～FFFFFEH），参见图3-2。

工作时，应用程序只能访问用户存储器空间，但是使用读数据表指令（TBLRD）和写数据表指令（TBLWT）可以读/写配置存储器中的器件ID号、用户ID号以及某些器件的配置位。配置存储器空间主要是留给Microchip公司进行器件测试以及存储一些重要的器件识别信息的。为了保持与数据空间的地址（下面将简要介绍）相兼容，相邻程序字之间的程序空间地址增量为2。

58

存储器的底端（地址为0）是一条GOTO指令，之后是应用程序起始代码的入口地址（保存在地址2处）。当处理器复位时就会执行这个双字指令，它将使处理器跳转到启动代码的最开始处以载入应用程序。接下来的地址4～7EH单元内保存中断向量表，表中记录所有中断服务程序的入口地址。在两个保留指令字地址80H和82H之后，接下来的存储器映射是备用中断向量表（地址84H～FEH），它的用途将在3.2节中介绍。

程序存储器的100H～17FFEH单元是用户Flash程序存储器，它能存放48k条指令（共占用144KB）。之后又是一个很大的保留存储器段（18000H～7FEFFEH），所有内容都为0。用

户存储器空间的最高4k字空间被数据EEPROM占据。

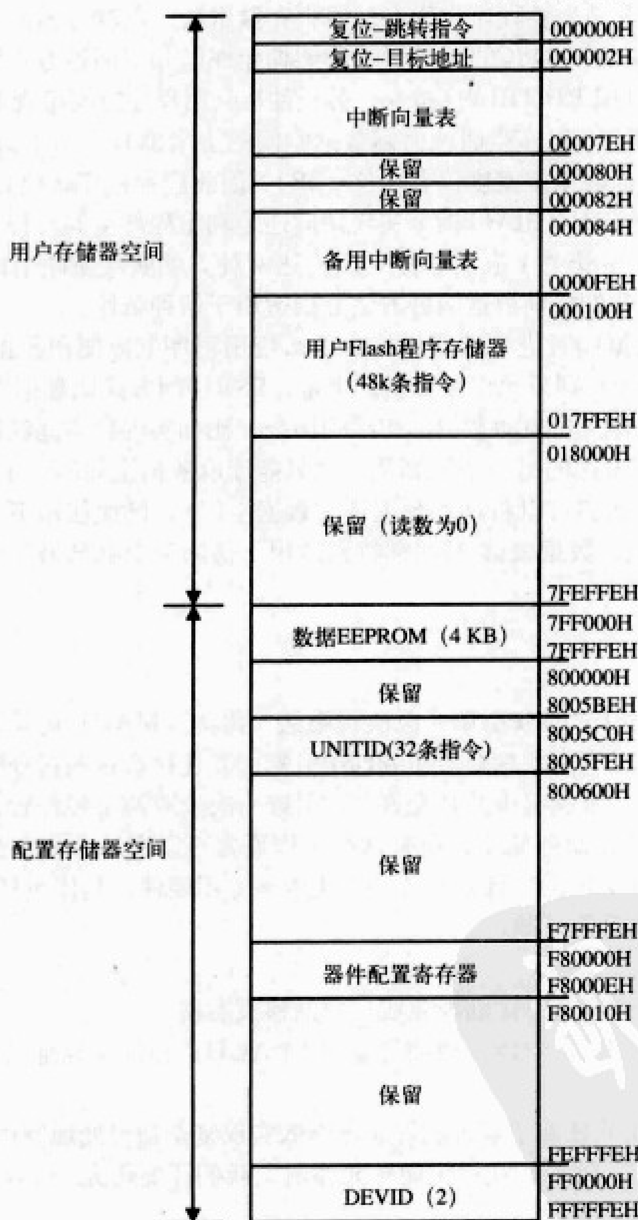


图3-2 dsPIC30F6014A的程序空间存储器映射

a. 从用户存储器空间访问配置存储器

通常情况下，应用程序仅工作在用户存储器空间内，但是，正如前面提到过的，应用程序还可以访问配置存储器空间内的部分空间，以便提取单元ID（32字）、器件ID（2字）以及读取或设置器件配置寄存器（16字）。为此，应用程序必须先将SFR中的数据表页面寄存器（TBLPAG）的第7位置“1”，然后使用读数据表指令（TBLRD）或者写数据表指令（TBLWT）向期望的地址单位进行读（从配置存储器读取数据）或者写（向配置存储器写入

数据）操作。

b. 将程序存储器映射到数据空间

DSP应用程序经常会在非易失性程序空间存储器中存储常数，以释放前面提到的普通数据空间内的RAM单元。dsPIC系列DSC的改进型哈佛架构允许以如下两种方式访问程序存储器：一种是前面提过的利用TBLRD/TBLWT指令，另一种是利用程序空间可视页（PSVPAG）寄存器将16k字（相当于32KB）程序空间映射到数据空间的上半部分。由于dsPICDSC的架构支持24位（3B）指令字，而数据字宽度仅有16位（2B），因此应用程序必须完成字节对齐，而所用方法则根据是利用TBLRD/TBLWT指令实现访问还是利用程序空间可视页寄存器实现存储器重映射而有所不同。尽管关于应用程序利用上述两种方法实现重映射的机理要留在3.1.3节中描述，但是在这里我们要明白这两种方法分别适用于何种条件。

59 ~ 60

如果需要对被访问的数据同时进行读写操作，那么应用程序就应使用数据表读/写的方法；而程序空间可视法（PSV）则只允许读数据。但是，PSV访问方式比数据表读/写方式更快，并且它特别适合于需要将常数与动态数据相乘的场合，比如实现数字滤波器或者快速傅里叶变换（FFT）。PSV访问方式的另一个限制是，它只能读取数据空间X，而实际中这种约束的影响并不严重，因为，我们可以将动态数据放在数据空间Y，已便使用下一节将要介绍的MAC类指令对其进行处理。数据表读/写法则特别适用于访问配置存储器空间的情况，该方法必不可少。

3.1.2 DSP引擎

dsPIC系列DSC集成了强大的DSP引擎，以实现乘法－累加（MAC）运算，这种运算是很多信号处理算法的基础。只有少数指令会用到DSP引擎，并且执行这些指令时还受到所处理的数据源的限制，但是这些限制很少并且允许DSP引擎一次读取两个操作数，然后对它们进行MAC运算，最后将结果存回存储器，所有这些工作通常可以在一个指令周期完成。为了达到这样的处理吞吐量，dsPIC的设计师在芯片上集成有专用硬件，具体包括：

- 1个17位×17位分数/整数乘法器；
- 2个40位累加器；
- 1个40级桶型移位器，它能在单周期内完成16位左移或右移；
- 2套地址产生单元（AGU），它能完成模寻址（2个AGU）和位反寻址（1个AGU）运算。

如果应用得当，这些专用的硬件子系统能够显著降低实现复杂信号处理算法的处理任务。图3-3是一个DSP引擎的框图。但是，在讨论硬件元件前，我们还要研究一下dsPIC系列DSC是如何表示数字的。

61

1. 数字的表示方式

dsPIC系列DSC可以处理有符号小数或者整数。有符号数采用标准的二进制补码格式，其中最高有效位（MSB）是符号位，后续各位都是2的幂次，具体见图3-4。

1个N位二进制补码整数可以表示的整数范围是 $-2^{N-1} \sim 2^{N-1}-1$。因此，16位二进制补码整数可以表示的整数范围是－32 768（0x8000）~32 767（0x7FFF），而32位可以表示的范围则是－2 147 483 648（0x80000000）~2 147 483 647（0x7FFFFFFF）。

62

当处理二进制补码整数时，关键要确保在对不同字长的数据进行加、减运算时对符号位

处理得当，否则就会发生计算错误。例如，如果我们对一个8位的二进制补码数-4（0xFC）不进行8位符号扩展就加上另一个16位数20（0x0014），那么得到的结果就是0x00FC+0x0014=0x0110（即十进制的272），根本不是正确值16（0x0010）。但是，如果在进行加法运算前对8位数进行符号扩展，那么得到的结果则是0xFFFC+0x0014=0x10010，如果保持16位字长不变，那么结果的最高有效位将被舍弃，于是得到0x0010。两个不同字长的补码数的加法运算法则是，扩展位数较短的那个数的符号位，使之与位数较长的那个数字长相等。注意，我们关心的是表示长度而不是我们所使用数据的大小。例如，如果将一个8位数与一个16位数相加，那么要先将这个8位有符号数扩展为16位有符号数，而不考虑这个8位或者16位数的大小。

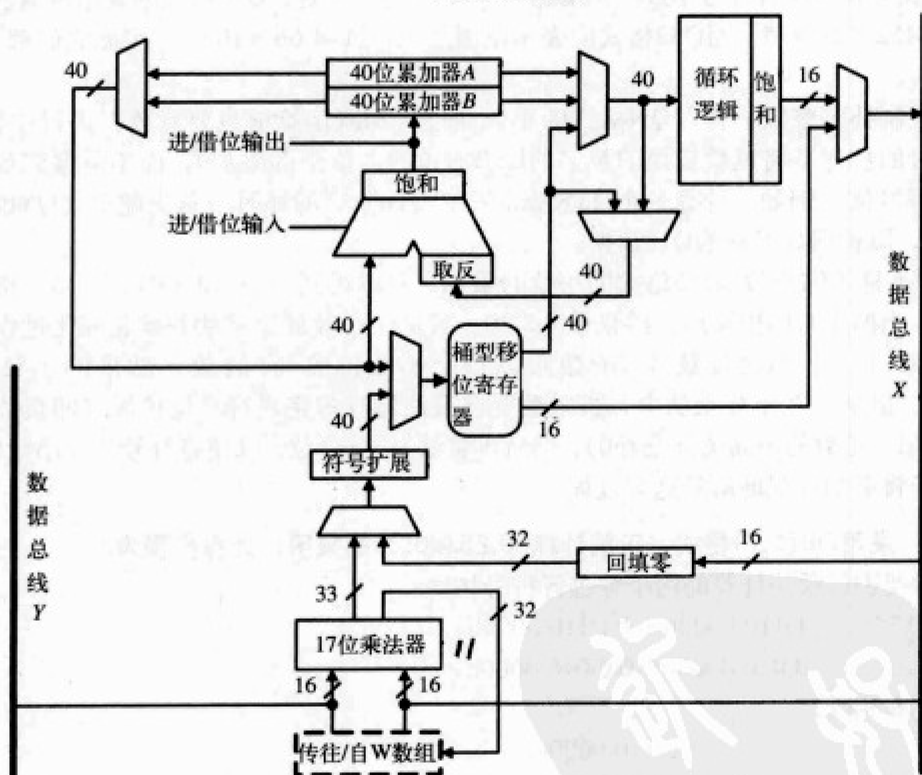


图3-3 dsPIC30F系列的DSP引擎的框图

-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

图3-4 16位有符号整数的补码表示

尽管整数的运算速度非常快，但是它在进行乘法运算时很容易溢出，因此通常需要应用系统另外引进软件来检查并处理溢出问题以防止得到错误的结果。但是，一个*M*位的整数与一个*N*位的整数相乘就得到一个（*N* + *M*）位的值，因此即便有一个很大的乘法器，也可能因为乘法次数太多而溢出。有一种避免溢出问题的方法是使用小数运算乘法，这样被乘数和乘数都被限制在-1～+1之间，乘法运算的结果也肯定在这个范围内，从而可以消除溢出问题（至少可以保证乘法运算不会溢出）。由于这种方法很容易实现，dsPIC系列DSC就可以采用

QN格式进行有符号小数运算。在QN格式中，数据被表示成一个二进制补码小数，其中最高有效位是符号位，紧随其后的是小数点（十进制）。QN格式中的N代表数据的总位数。比如Q16格式的数就有16位，而Q32格式的数就有32位。该符号还代表所采用的数据表示是1.(N–1)格式，其中(N–1)项代表尾数（数据的小数部分）的位数。采用这种表示方法时，Q16数据就采用1.15格式，而Q32数据则采用1.31格式。图3-5是一个采用Q16（或者1.15）格式表示的数据例子。

-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

63

图3-5 有符号小数的Q16（或1.15）格式表示

QN格式可表示的有符号小数的范围是$-1\sim(1-2^{1-N})$。于是，Q16格式的表示范围就是–1～0.999 969 482 421 875，而Q32格式的表示范围是$-1\sim(1-4.66\times10^{-10})$，可见它们都在–1～1这个范围内。

和二进制补码整数一样，QN格式的小数在进行加减运算前也要对数据进行位数扩展。但是，此时的扩展是将尾数较短的数字的LSB右边的各位全部添加0，而不是像整数运算时那样扩展符号位。例如，计算一个Q8数据和一个Q16数据的和时，首先就要在Q8数据右侧添加8个零，以扩展成对应的Q16数据。

计算$M.N$格式的小数与$R.S$格式的小数的乘积，可以得到一个（$M+R$）.（$N+S$）格式的结果。于是，当两个1.15相乘后，结果就是2.30。可见，小数部分乘法的字长变化过程与整数乘法的类似［一个小数位数为N的数乘以一个小数位数为S的数，结果的小数位数为（$N+S$）］。但是，在小数乘法中，被乘数要在乘法运算前完成符号位扩展（即负数的最高位之前全补1，正数的最高位前全补0），然后再将结果左移1位，以便将符号位和小数位对齐。下面的例子将更加直观地解释这个过程。

例3-1：采用Q8(1.7)格式，正确计算–0.25和0.25的乘积，计算步骤为：

（1）对要进行乘法计算的两个数进行符号拓展。

–0.25:　1.110 0000b→1 1.110 0000b

0.25　0.010 0000b→0 0.010 0000b

（2）两数相乘。

```
             1 1.110 0000b
           × 0 0.010 0000b
        11 1100 0000 0000b
       000 0000 0000 0000b
      0000 0000 0000 0000b
    0 0000 0000 0000 0000b
    0 0001 1110 0000 0000b
```

64

（3）结果左移1位。

0011 1100 0000 0000b

（4）将小数点放在倒数第13位的前面。

001.1 1100 0000 0000b = – 0.0625 decimal

2. 硬件乘法器

dsPIC系列DSC具有17位×17位的硬件乘法器，能实现两个有符号16位整数或小数的乘法运算。需要注意的是，除非特殊情况，否则整数运算和小数的乘法运算过程是相同的，二者唯一的差别是对结果的解释不同。例如，如果将两个16位二进制补码整数0x2001和0x17F0相加，那么结果是0x37F1。如果是同样的两个有符号Q16数据（0x2001和0x17F0）相加，那么结果也是0x37F1。但是二者的区别在于，如果将0x37F1看作16位二进制补码整数，那么就代表十进制数14 321。如果将它看作Q16数据，其值近似为0.437 042 236。

如前文所述，二进制补码型有符号小数的乘法需要对乘数的符号位扩展1位，因此将两个Q16数据相乘时需要使用17位×17位的乘法器。对于两个Q16格式的乘数中有一个是0x8000（–1）的特殊情况，硬件乘法器会自动将数据更正为最接近于+1的0x7FFFFFFF。如果硬件不这样处理误差，那么会导致溢出并得到错误的结果：(–1)×(–1)=0。尽管这种更正会带来一点小小的误差，但是这比乘法器溢出的影响要小得多。

由于DSP引擎处理整数和小数乘法的方式不同，应用程序必须指明所采用的数据表示方式。这可以通过配置CORCON（内核配置）特殊功能寄存器的IF位来实现。当该位被清零时（设置成“0”），乘法器工作在小数乘法模式；当该位被置位时（设置成“1”），乘法器工作在标准整数乘法模式。在小数乘法模式下，乘法器会自动完成符号位扩展和为了确保小数点对齐所必需的左移操作。如果应用程序在使用MAC或者MPY指令（它们都属于DSP乘法指令）处理小数前，未将IF标志位清零，那么应用程序就得自己完成左移，否则就会得到错误的结果。

而标准MCU的乘法指令（各种MUL指令）也使用同样的乘法器，可以处理8位或16位整数（有符号数或者无符号数都可以）运算。当两个操作数都是8位时，结果就是16位的；当两个操作数都是16位时，结果就是32位的。

65

3. 双40位累加器

乘法器的输出会被送往数据累加器，它包括一个带有符号扩展逻辑的40位加法器/减法器和两个40位累加器（分别叫累加器*A*和累加器*B*），这两个累加器可以存放累加运算时的源数据或者结果数据。对于某些指令，比如ADD（累加器加法运算）和LAC（将数据载入累加器），还可以在进行累加操作前利用桶型移位器对数据进行缩放处理，这可以方便地实现数据归一化。当进行复杂的数字运算时，配备两个累加器是非常有用的，这也是常用DSP需求的一种。

可能有人会问，为什么乘法输出只有32位，而数据累加器却有40位呢？答案是累加器越宽，可以提供的运算“空间”就越大，从而能够防止累加器在进行多次累加（在多抽头滤波器中常常遇到）后出现溢出等问题。请回忆一下，我们采用小数计算乘法的原因就是为了避免乘法运算出现溢出，为此，我们将输入操作数的大小限制在–1～+1，这样就能保证计算结果也始终界于–1～+1。但遗憾的是，我们无法限制加法或减法运算的输出范围，因为两个小数的和很可能超出–1～+1这个范围。因此防止溢出的唯一方法就是在小数点左侧提供足够的“保护”位，保护位越多越好。对于dsPIC系列DSC来说，芯片设计师设计了8个保护位（等于40位累加器与32位输入数据的字长之差），从而允许对任何一侧输入数据连续加上256次满幅值（如两个–1值相加或两个+1值相加）也不会溢出。实际中，根据实际累加的数据大小，这8个保护位可以允许累加器在溢出之前完成更多次的累加运算，或者绝不会溢出。

由于计算溢出是个大问题，因此dsPIC系列DSC为每个累加器提供了2个标志位，以便应用程序检查溢出。这些标志位位于CORCON寄存器内，用于指示是否发生下列情况。

（1）累加器的第39位已经溢出（由SA或SB位指示，“1”有效）。

（2）累加器已经溢出到保护位内（由OA或OB位指示，“1”有效）。

dsPIC30F系列的参考手册明确地将上述第一个条件描述成“致命性溢出”。因为一旦发生这种情况，累加器的符号位就被破坏了。第二个条件则不太严重，但是它表明了潜在的问题：随后的累加可能导致致命性溢出。

66

发生致命性溢出后会引起严重问题，因为此时的运算结果可能从正数变成了负数，或者相反。这表现为数学上的不连续现象，从而破坏了大多数数字信号处理（当然也包括本书所讨论的DSP在内）的第一准则——可以将它们视为线性系统进行数字建模。大多数微处理器都很难检查和纠正致命性溢出，并且代码在时间和空间上的花销都很大。但是dsPIC的设计者配备了专用的可配置逻辑，它不仅能够检测出致命性溢出，而且还能在不增加软件的情况下由数据累加器自如地应对这种情况。通过适当配置，dsPIC系列DSC可以将致命性溢出转换为最小负值或者最大正值（根据所检测到的溢出情况而定），即实现了所谓的*累加器饱和*功能。在这方面，dsPIC系列DSC模仿了模拟乘法器处理溢出的方法，该乘法器在溢出后会使输出饱和值而不会翻转。dsPIC系列DSC的饱和特性可以根据应用需要定制成普通范围（32位）或扩展范围（40位）下的溢出饱和。

dsPIC系列DSC能够灵活地处理溢出检测标志和两种不同的溢出。这两种标志（OA/OB和SA/SB）仅在数据每次通过加法器/减法器时被修改，但是SA/SB标志只能由软件清零，而OA/OB标志则可在溢出条件消失后由硬件自动清除。此外，如果中断控制寄存器（INTCON1）中的溢出中断使能位（OVATEN/OVBTEN）被置位，那么对任何一种溢出标志位置位后都会产生运算警告中断（可选的）。这使得用户能够立即处理溢出错误，比如可以减小系统增益以消除该问题。

为了进一步减小用于检测和处理溢出条件的预置程序，dsPIC系列DSC还有两个标志位OAB和SAB，它们分别是OA和OB、SA和SB位的逻辑或。这两个标志位位于状态寄存器（SR）内，从而允许应用程序迅速检测出累加器溢出，这在处理复杂数学运算的场合很有用。

4. 40位桶型移位寄存器

40位桶型移位寄存器能够迅速（单周期）将数据右移15位或者左移16位。该特性对特定范围内的数据归一化处理以及对齐来自通信端口的串行数据流非常有用。通过X总线可以将待移位数据送入移位寄存器。其中16～31位是需要右移的数据，而0～15位则是需要左移的数据。

67

3.1.3 寻址方式和地址发生单元

dsPIC系列DSC具有高数据吞吐量的关键原因之一是，它具有很多灵活的寻址方式，从而能高效地访问并处理数据。该芯片不仅具备多种寻址方式，而且还采用基于硬件的地址发生单元（AGU）实现无开销的求模运算以及位反寻址功能，从而能够显著降低复杂信号处理算法的软件开销。精心利用AGU特性以及各种寻址方式能够显著精简代码并缩短算法的执行时间。

dsPIC系列DSC支持4种基本数据寻址方式，其中有些还会在某些指令下有所扩展。

（1）页面寄存器或存储器直接寻址方式。

（2）寄存器直接寻址方式。

（3）寄存器间接寻址方式。

（4）立即数寻址方式。

页面寄存器或存储器直接寻址方式在指令中嵌入了13位绝对存储器地址。由于这种方式只允许使用13位地址，因此它只能访问低8KB数据空间。这种寻址方式最常用于访问特殊功能寄存器（SFR），它们正位于低2KB数据空间。请注意这种寻址方式无法访问SRAM的高2KB单元。

寄存器直接寻址方式允许直接访问W寄存器序列中的所有寄存器，支持这种寻址方式的指令的操作数据源和结果目标寄存器都是W寄存器（比如W1和W4等）。这种方式特别适用于具有$Z=X+Y$形式的三操作数指令，其中X、Y和Z都是W寄存器，允许的运算包括加法、减法或者位运算。请注意，在三操作数指令中，两个源操作数必须位于不同的寄存器中，而目标寄存器则可以指向任何一个W寄存器。

通过实现寄存器间接寻址方式，该芯片能够轻松地支持高级语言，特别是含有数据指针的C语言。寄存器间接寻址方式的标志是被访问数据的地址保存在一个16位的W寄存器中。这与其他寻址方式相比有两个优点：一是它允许应用程序访问更大的数据空间（可达64KB，而页面寄存器寻址方式只有8KB，而寄存器直接寻址方式只有32B）；另一个是它允许应用程序实时修改待访问数据的地址。

68

在这些基本的寻址方式中，寄存器间接寻址方式最为灵活，它使应用程序在单周期指令内递增或递减源操作数寄存器和/或目标结果寄存器的数据。这种递增/递减操作可以在提取被寻址的数据之前进行，也可以在提取之后进行，从而能够用于产生各种强大的循环结构。此外，有些指令还允许地址寄存器的内容包含寄存器偏移量（例如，由地址寄存器的内容加上另一个寄存器的内容后得到目标地址）或者立即数偏移量（例如，由地址寄存器的内容加上固定的数字得到目标地址）。最后介绍的这两种用法使得应用程序能够高效地访问固定数据表或者结构体中的数据。

最后，立即数寻址方式则是在指令中嵌入数据，数据的位数大小与所用指令有关。

3.2 中断的结构

中断是在指令执行过程中的异步变化。它可能是由外部事件（比如，硬件信号状态发生改变）引起的，也可能是由内部条件引起的（比如内部定时器溢出）。从技术上看，dsPIC系列DSC将中断分为硬中断和陷阱。硬中断是由正常应用程序中的预定处理事件产生的，而陷阱（或者是处理器异常）是负责处理错误的，它们在处理器正常运行时不会出现。实际中，中断和陷阱的处理方法是一样的，因此这里的讨论将统一称之为“中断”。

dsPIC系列DSC共支持45个中断源、4种陷阱以及可以使处理器复位的6个条件。由于大多数应用只使用其中的少数异常处理服务程序，因此dsPIC系列DSC的中断必须分别由全局中断使能标志使能，而全局中断使能标志位则允许程序在单周期内迅速地关断所有的中断。当发生中断事件时，如果相应的中断使能位和全局中断使能位都已被置位（使能），那么处理器就会把它当前运行处的程序地址存储在中断栈中，并且从中断向量表（Interrupt Vector Table，IVT）中对应的入口处提取中断处理函数的地址（也就是所谓的*中断向量*）。中断向量表的地址位于程序空间的000004H～00007FH地址段。提取了中断程序的地址后，dsPIC

系列DSC就会跳转到该地址并且开始执行代码，直到它遇到中断返回指令（Return from Interrupt Enable，RTFIE），之后它会提取并恢复最近存储在中断栈中的地址，然后继续执行程序。

69

为简单起见，前面的讨论忽略了dsPIC系列DSC处理中断的一个关键内容，就是它共有7个中断优先级。其中第0级的优先级别最低，而第6级的最高。这意味着这些中断具有内建的层次，有些中断更重要一些。在中断向量表中的中断源地址越低则优先级越高，它们的先后顺序由它们出现在中断向量表中的自然顺序决定。由于dsPIC的设计师意识到并非所有的应用都要求不同中断具有相同的优先级，因此器件允许应用程序在运行时升高或者降低每个中断的优先级。

中断处理的关键是从中断发生到处理器实际开始响应之间的时间，也就是所谓的中断等待时间。引入中断等待时间的目的有两个，一是使启动中断处理的时间尽可能短，另一个是使中断等待时间要尽可能地稳定，以便应用程序中的每个处理部分能可靠地处理它。在dsPIC系列DSC中，这两个目的完全达到，并且从中断请求发生（即中断源生效的时刻）到中断服务例程（ISR）真正开始执行的时间等于固定的5个指令周期。但是必须注意，该中断等待时间是在假设当前没有处理同级或者更高优先级的中断的情况下得到的；如果处理器正在处理其他的同级或者更高优先级的中断，那么就要等到它们的ISR运行完毕后才能处理新中断，这样，中断等待时间就会大于5个指令周期。

影子寄存器

前面的讨论还遗漏了dsPIC系列DSC的另一个关键性能增强特性，使用影子寄存器可以在出现中断或者其他异常时实现快速的上下文切换。当发生中断时，应用程序可以将当前状态寄存器中的部分标志位以及TBLPAG、PSVPAG和W0~W14寄存器的内容存储到影子寄存器中，而这一切只需调用一个单周期指令PUSH.S即可完成。当使用影子寄存器的中断服务程序执行完毕时，ISR必须调用POP.S指令将存放在影子寄存器中的数据放回标准寄存器。

由于影子寄存器只有一级深度，因此应用程序绝不能连续调用PUSH.S或者POP.S指令两次，否则原始寄存器的内容就会丢失，这一点至关重要。这意味着如果优先级较高的中断服务程序使用影子寄存器，那么优先级较低的中断服务程序就不能使用同一个影子寄存器，否则当处理某个低优先级的中断时，一旦发生优先级更高的中断，就会导致影子寄存器的内容被重写。

70

3.3 片上外围设备

dsPIC系列DSC最强大的一点是它显著提高了传感器应用系统的集成度。根据具体情况，1枚单片dsPIC DSC集成的片上外围设备模块能够直接数字化模拟输入信号或者从外部A/D转换器读取数字值，还含有多个可以自由运行和事件驱动的定时器，并支持多种工业标准通信协议。由于dsPIC的架构允许设计师指定每个外围设备在运行时的工作优先级，因此能够针对特殊应用来优化芯片的处理和吞吐能力。

在下面有关各种外围设备模块的讨论中，我们关心的是明确dsPIC数据手册中的信息，并且制定检查表，以便设计师能够正确地使用每个模块。此外，还包括Microchip公司提供的其他资源，比如教学和应用笔记。目的只有一个，就是使读者能通过该芯片大量原始资料

建立一个连贯的知识体系。

3.3.1 数据采集外围设备

在任何智能传感器系统中，DSP信号链[1]的“数字”部分都开始于用一种所谓的模数转换器（也称为A/D或者ADC）对模拟输入信号进行数字化。dsPIC系列DSC支持多种方式的信号数字化：

- 通过片上的多通道ADC采集；
- 用外部的音频编码/解码器（codec）采集，并由dsPIC的数据转换接口（DCI）读入采集结果；
- 由通过串行外围设备接口（SPI）或者内部集成电路（I^2C）端口控制的外置ADC实现采集。

本书中，我们将着重介绍第一种方式——利用片上ADC实现数据采集。

1. 模数转换器

对输入信号进行数字化的最常用也是最经济的方法恐怕就是使用dsPIC系列DSC的内部ADC了。正如这里所说的，所有dsPIC30F系列器件都支持6～16通道ADC输入，根据器件型号不同，分辨率为10位或12位。具有10位分辨率的器件通常用于电机控制或者视频处理等分辨率较低的应用（它只有1024个量化等级，而12位ADC则有4096个量化等级），而具有12位分辨率的则适用于传感器市场。本书采用的是dsPIC30F6014A，它集成有16通道12位分辨率ADC。

71

为了使读者能对不同分辨率对实际应用的影响有一个直观的感受，下面介绍两个实例，一个是检测驾驶控制系统中方向盘的转动位置，另一个是测量注模机模具内部的压力。在方向盘位置检查系统中，我们假设能利用满分辨率的ADC测量360°的转动，并确定方向盘的转动位置。在这种情况下，利用10位ADC就能测量出方向盘大约三分之一度的转动（$360°/2^{10}$级$=360°/1024= 0.35°$/级），而12位ADC则能分辨出不超过十分之一度的转动。因此上述任何一种器件都能很好地满足大多数驾控系统的需要，完全能达到人的感知要求（当然，假设是人工驾驶）。

但是当监视大尺度信号时情况就不那么明朗了。比如注模机内的压力，通常在0～30 000psi[2]之间变化。于是10位ADC每个量化等级对应约30psi，而12位ADC则对应7psi。这看起来影响不大，但是如果注模机在注模过程中的关键阶段要用到这个压力值，以便打开阀门通入更多塑料的话，那么不同的分辨率就会要么得到稳定的、有盈利的注模过程，且废品率低，要么得到不稳定的、无效益的生产过程，且废品率很高。

Microchip公司的Richard Fischer在他组织的基于dsPIC的ADC模块的在线研讨会[3]中指出，ADC每次将模拟输入信号转换为对应的数字值的过程可以分为两步。第一步，ADC采样模拟输入信号，并且将模拟采样值存储在采样－保持放大器中，该过程被称为获取。Richard指出，捕获模拟样值就像给模拟波形拍照一样，而获取时间则类似于照相的曝光时间。每次获取允许的时间由具体的应用决定，但是必须保证在获取时间内能准确地捕获信号，这就像照相时必须保证在相机的快门时间内能有足够的光照通向底片（或者是新一代数码相

---

① 信号链是指信号从输入到输出经过的处理过程。它通常包括模拟和数字成分，至少在采用数字信号处理技术的系统中是这样的。

② psi=磅/平方英寸。

③ dsPIC30F 12-bit ADC Module，Part 1 of 2，第6页。Microchip公司的网站上有该研讨会资料。

机的CCD阵列)，这样才能拍出好照片。

一旦信号被捕获，ADC就开始第二步处理：将采样的模拟信号转换为对应的数字值，该过程被称为转换①。图3-6显示了采样时间（获取时间）、A/D转换时间及完整的转换周期时间。根据ADC模块的配置情况，转换结果可以表示为4种不同的整数或小数格式，从而允许应用程序根据自己的算法选用最合适的数据格式。

72

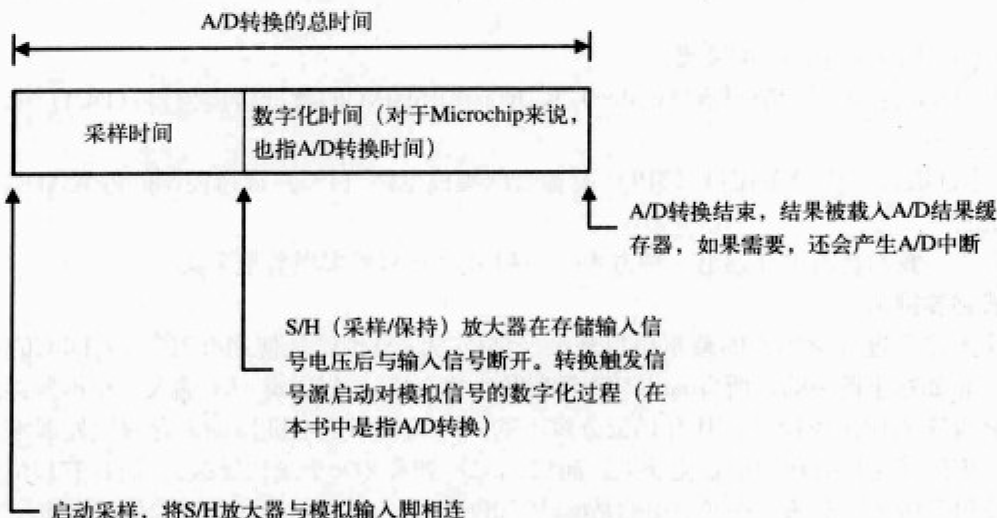


图3-6　A/D转换的时序

dsPIC应用系统可以采用轮询（或手动）方式或者自动转换方式采样数据。其中，轮询方式是由代码直接启动每次采样操作，而自动转换方式则在完成ADC模块初始配置后，由硬件反复实施获取与转换，并仅在指定的通道转换结束后向处理器发中断。大多数应用更愿意采用中断驱动方式，因为它能保证时间固定地采样数据——即相邻两次采样的时间非常稳定。这种时序稳定性非常重要，因为大多数DSP算法都假设采样频率为常数，意外的抖动②会导致错误的结果，但是如果更严格地控制采样周期就可能看不出这种错误。轮询方式通常在调试硬件接口时使用，它能以异步方式“现场查看”采样结果（例如，在执行“引擎检查”操作时查看汽车内的油量），或者在采样速率非常低无法由ADC模块的硬件处理时使用。

dsPIC系列DSC采用单个逐次逼近型多路输入通道复用的A/D来实现数字化。根据ADC分辨率和器件型号，使用了1个、2个或4个采样保持（S/H）放大器实现多输入通道复用，具体见图3-7。只有10位ADC支持多路S/H放大器，而12位ADC则只能使用一个S/H电路。乍一看，S/H放大器可能没什么重要作用，因为用于转换输出的转换电路仍然只有一套，并不能提高转换效率。但是这些S/H放大器能实现真正的同步采样，这一点非常重要。通常，该特性对特定的设计有好处，但不是必备的。在某些需要它的场合，dsPIC系列DSC的片上多个S/H放大器就体现出极大的优势。

73

内部A/D在满幅供电电压下的最大采样速率不足200ksps（每秒采样200 000次），但是其

① 实际上，转换一词不仅指转换采样保持放大器上模拟信号（本节就是这个意思），而且还指整个采样/转换周期。在大多数情况下，从上下文可以清楚地看出它的意思。

② 抖动是指与采样周期的标称值的偏差。比如，如果某系统采样周期的标称值为2ms，而实际的采用周期在1.8~2.1ms变化，那么该系统的抖动就为0.3ms或15%，说明其性能不太好。

采样带宽可以延伸到所有通道，因此当需要采样所有通道时，每个通道的最大采样速率只有12 500sps（200 000sps/16通道=12 500sps/通道）。当然，实际采样速率可能还要减小，以保证有足够的处理时间来完成应用系统的信号调理和算法分析。此外，采样速率还受到芯片供电电压的影响；当该芯片的供电电压为4.5～5.5V时，实际采样速率不足200ksps。如果供电电压更低，那么最大采样速率就会下降到100ksps。

12位高速A/D的结构框图

图3-7 dsPIC的模数转换器电路

如图3-7所示，该ADC电路具有很大的灵活性，但是这种灵活性要求应用程序完成必要的配置后才能正常使用。ADC模块具有6个16位状态与控制寄存器，以及1个用于存放转换结果的16字双端口只读缓冲寄存器（ADCBUF0～ADCBUFF）。

74

在下面的讨论中，我们将按照完成采样任务的工作顺序来剖析ADC的配置过程，而不是简单地列出各寄存器的每一位是何作用。希望这种方式能帮助读者更好地理解整个处理过程，而不只是记住一堆寄存器配置的列表。当然，为了完整起见，我们在最后也会给出这种列表。

在配置ADC模块前，我们首先需要明确以下内容。

（1）需要将哪些I/O管脚用作模拟输入端口，哪些用作数字输入端口，哪些用作数字输出端口？

（2）需要多高的采样速率？

（3）是否需要采用交错采样，或者只需在给定的采样周期内完成对所有信号的转换后产生一个中断？为了达到处理吞吐量要求，是否需要多路复用结果缓冲器？

（4）转换结果要表示成何种格式？

一旦回答了上述问题，我们就能真正开始配置ADC模块了。

a. 配置信号通路

显然，配置ADC模块的第一步就是确定将哪些管脚作为模拟输入端口、数字输入端口或

者数字输出端口。尽管也可以在程序运行时更改配置，但是为了简单起见，这里假设仅在执行程序之初配置I/O端口的类型和方向。有些应用需要在程序运行后改变信号类型和方向，但是，我们最好在中断被禁止之后再按以下步骤进行配置。如果由于认为信号配置不同而运行程序，绝不允许改变I/O信号的类型或者方向。

b. 配置I/O端口管脚

作为Microchip产品的标准配置，应用程序可以通过TRISx寄存器指定I/O端口信号的方向，其中x表示某个端口（TRISA对应PORTA，TRISB对应PORTB，等等）。16位TRISx寄存器与16位PORTx寄存器是逐位对应的。若要将某个芯片配置成输入（无论是数字或是模拟），应用程序就要将TRISx的对应位置“1”；若要将该信号配置位输出，那么应用程序就要将TRISx的对应为置“0”。为便于记忆，可以将“0”看作0utput，而将“1”看作1nput①。图3-8是通用TRISx寄存器的映射定义。 [75]

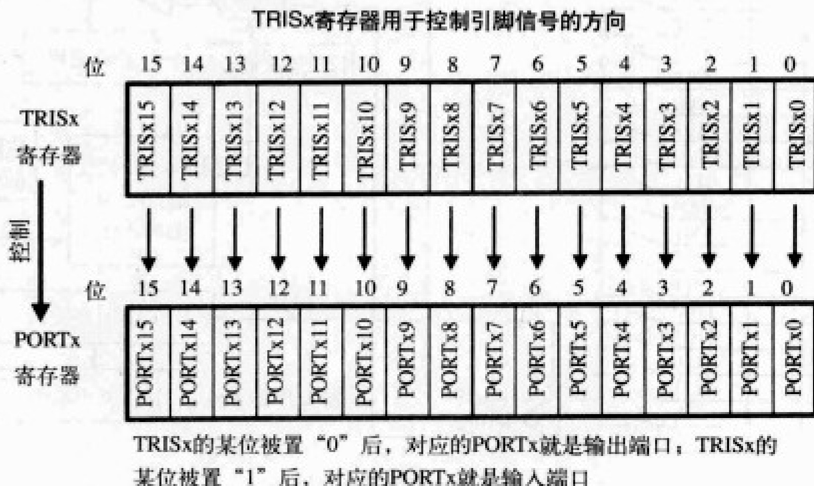


图3-8 通用TRISx寄存器的位映射

由于dsPIC30F6014A的所有模拟输入引脚（数据手册中记作AN0～AN15）都位于端口B上，因此为了正确配置模拟输入引脚，只需关注寄存器TRISB即可。如果希望对端口B上某路信号进行数字化，那么就必须将TRISB中的对应位置“1”。例如，如果希望数字化AN0、AN1和AN8上的信号，同时又要求端口B<4>和端口B<6>作为数字输入端口，而其余引脚都是数字输出端口，那么就要向TRISB寄存器写0x0153［由0x0103（模拟输入端口）或0x0050（数字输入端口）计算得到］。

确定了信号方向之后，再确定输入信号是数字的还是模拟的。这可以通过配置16位A/D引脚配置寄存器（ADDCFG）完成。该寄存器中的各位对应于模拟输入端口（AN0～AN15）。如果某位被置“1”，那么它就是数字输入端口；如果某位被清“0”，那么它就是模拟输入端口。该寄存器的位映射定义参见图3-9。

继续前面的例子，我们需要将ADPCFG寄存器配置为0xFEFC，只将AN0、AN1及AN8对应位清零（它们在本例中只用作模拟输入端口）。 [76]

如果应用程序将某个TRISB引脚配置成数字输出，但是又同时在ADPCFG寄存器中将它

① 这两个“单词”的首字母就是数字（前提是可以将数字与字母的组合称为一个“单词”）。关键是记住转换关系，而不是它破坏了英语规则。尽管这种助记法并不是作者的原创，但是实践证明它的确很管用。

配置为模拟输入引脚，那么会发生什么情况呢？在这种情况下，引脚会被用作数字输出端口，但是同时我们又能将该引脚上的电压数字化，并从ADC模块读出数字化结果。该特性在强调安全性的应用中很有用，因为它能使处理器确定某个数字输出引脚的状态是否正确，或者是否短接到地或V_{DD}。



图3-9 AD引脚配置寄存器的位映射

尽管下面将介绍配置ADC模块本身，但是在大多数应用中，设计者更希望先配置其他I/O端口的方向和状态（即端口A、C等），以便能尽快将硬件配置成可知的安全状态。特别需要注意的一点是，在某个模拟输入引脚上施加的模拟电压可能使得其输入缓冲器承受比额定值更大的电流（这是从技术性角度说明它可能会损坏芯片）。

c. 选择参考电源

配置了信号方向与类型后，下面我们将选择数字化时需要使用的参考电压源。为简化设计，转换范围通常是整个供电电压的范围（即从$AV_{SS}\sim AV_{DD}$）。但是，为了提高测量的分辨率，还可以将转换参考电压限制在$V_{REF-}\sim V_{REF+}$的范围内。这样可以显著提高芯片测量限幅输入信号的能力，因为只需转换感兴趣的电压范围内的信号即可，而无需转换整个供电电压范围内的信号。例如，如果设计师知道输入信号的范围是0.5～1.5V，那么他可以将参考电压的上下限调整为上述两个值，以便ADC能够分辨0.24mV的电压增量（$(V_{REF+}-V_{REF-})/2^{12}$级=(1.5V−0.5V)/4096级=0.24mV/级）。但是如果将V_{REF-}和V_{REF+}分别设为0V和5V，就像在标准的5V供电系统中一样，那么分辨率就会降低，变成原来的5倍（即1.2mV/级）。这可能看起来不是很糟糕，但是对于低电压输入信号，比如热电偶的输出信号，差别就很大。

77

此外，还可以将ADC模块配置成转换某路输入相对与另一路输入的电压差，从而实现差分测量。但是这种方式也有限制，它要求转换器的输入信号必须是单极型的，即要求参考输入电压必须小于或等于非参考输入信号。尽管存在这样的约束，但是差分工作方式还是能够有效降低输入信号中的共模噪声①。

① 共模噪声是一种同时出现在系统差分输入端的电气噪声。它常出现于双绞线，其导线导体是内部互相绞合的，因此任何辐射性噪声一旦耦合进入其中一根导线，那么也会同时耦合进入另一根导线。理论上说，由于差分接收器仅关心两根导线上的电压降，而任何同时出现在两根导线上的电压（包括噪声电压）在相减后都被抵消，因此差分信号对这种噪声有免疫功能。实际中，并不完全如此，但是这仍然是一种去除噪声影响的有效方法。

我们还记得，参考电压上、下限对应于模拟输入电压的范围，它们将被映射到10位或12位ADC输出值。令参考电压范围尽可能与模拟信号相匹配的目的是使转换结果具有最高的分辨率。对于dsPIC系列DSC而言，其参考电压共有四种组合，即选择AV_{SS}或V_{REFL}作为参考电压下限，而选择AV_{DD}或V_{REFH}作为参考电压上限。

配置参考电压源的方法很简单，只需设置A/D控制寄存器2（ADCON2）中的电压配置位（VCFG<2:0>）即可。尽管VCFG有3位，理论上可以产生8种参考电压配置，但是当其最高位（VCFG<2>）被置为“1”时，参考电压上限始终为AV_{DD}而下限始终为AV_{SS}。图3-10中的表格显示了参考电压配置与VCFG值的对应情况，而图3-11则给出了ADCON2寄存器中VCFG的位定义。

通过ADCON2中的VCFG位选择ADC的参考电压

VCFG<2:0>	V_{REFH}	V_{REFL}
000	AV_{DD}	AV_{SS}
001	V_{REF+}	AV_{SS}
010	AV_{DD}	V_{REF-}
011	V_{REF+}	V_{REF-}
1xx	AV_{DD}	AV_{SS}

V_{REFH} = A/D的转换电压上限

V_{REFL} = A/D的转换电压下限

V_{REF+} = 外部参考电压的上限

V_{REF-} = 外部参考电压的下限

AV_{DD} = 模拟正电源轨

AV_{SS} = 模拟负电源轨（地）

图3-10 ADC参考电压的配置值

ADCON2中的VCFG位映射

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCON2寄存器	VCFG2	VCFG1	VCFG0	-	-	CSCNA	-	-	BUFS	-	SMPI3	SMPI2	SMPI1	SMPI0	BUFM	ALTS

图3-11 ADCON2寄存器中的VCFG位映射

d. 选择需要数字化的模拟输入信号

为了减少不必要的预处理，应用程序可以指定某几个模拟输入通道进行转换、转换的次序（至少也分等级）以及产生中断前完成的转换次数等。由于只需转换少数几个ADC通道，因此能显著降低应用程序的处理时间，而硬件也无需花费时间处理那些不使用的通道所产生的转换结果。在所有采样方式中，数字化结果都会按照采样顺序被依次存储在16字ADCBUF缓冲器中，从ADCBUF0开始，直到由ADCON2寄存器中的每个中断采样次数（Samples Per Interrupt，SMPI）值指定的采样数值的对应缓冲器，然后在该位置的下一个采样结果又被存储在ADCBUF0。这样，应用程序的ADC中断服务程序必须保证在单个采样周期内能从ADCBUF中提取并处理完所有的采样值，如果无法满足这个时间要求，那么存储在

78

ADCBUF0中的数据就会丢失（其实是被新一次的采样值覆盖）。

ADC模块可以按照两种用户可配置的通道方式进行信号转换，甚至还能将这两种方式组合起来。第一种是所谓的*通道交错或通道轮流式采样*，它先数字化某个输入通道（可以想象成是MuxA输入，并将数字化值A存入ADCBUF0；然后再数字化第2个[通常是不同的输出通道(记作MuxB输入)]，并将结果存入ADCBUF1。就这样以交替方式连续采样（先采样MuxA输入，然后采样MuxB输入），直到将SMPI转换都存入ADCBUF，最后ADC模块产生一个中断信号（假设中断已使能）。产生中断后，下一次的采样值又被存储在ADCBUF0中，然后重复上述处理。

79

为了启动基本的交替采样（即轮流采样两个输入通道），应用程序必须将ADCON2中的轮流采样（ALTS）位置"1"。它还必须将A/D通道选择寄存器（ADCHS）的4位通道0选择A（CH0SA）区域和通道0选择B（CH0SB）区域设置为基于0的MuxA和MuxB输入的变址，进而指定Mux A和Mux B分别为相应的采样通道。与配置ADCHS一样，应用程序还可以通过对通道0负输入选择A标志（CH0NA）和通道0负输入选择B标志（CH0NB）置位以便将负输入S/H电路接入Mux A和Mux B。而清除CH0NA或者CH0NB标志则会选择V_{REF-}作为对应Mux输入信号的负输入参考。这样，ADC模块就支持单极型差分方式输入，要求是AN1上电压必须始终不超过所选择的Mux A或Mux B通道上的电压。

第二种采样方式是所谓的*通道扫描方式*，它能获得一组半连续通道的采样数据，并能在规定的采样次数完成后产生中断。"半连续"一词表示以固定的连续顺序转换各通道的信号（从低序号通道开始直到最高序号通道），如果不需要某个通道的数据，可以将该通道从采样序列中去除。和交错式采样一样，从ADCBUFO开始，数字化结果也会存储在ADCBUF中，处理器产生一个中断以便数据能从缓冲器传至应用程序中。中断后新的采样值又会被存储在ADCBUF中，并且是从ADCBUF0开始的，即便整个ADCBUF之前还未被填满也是如此。

最后，dsPIC系列DSC还支持交错式采样和通道扫描式采样的组合，条件是只能在MuxA部分进行通道扫描采样（即Mux B只有单个输入）。这适用于扫描一组动态输入信号和一个参考输入。(比如冷结补偿[①]温度的情况。)

下面三个简单的例子有助于理解交错式采样、通道扫描式采样以及交错与通道扫描组合式采样这些概念。

例子1：基本的交错式采样

80

假设要轮流采样通道2（即AN2）和通道14（AN14），其中通道2的输入以V_{REF-}为参考，而通道14的输入则以AN1为参考（单极性差分方式），我们希望ADC模块在对每个通道完成3次采样后（每次中断共计完成采样6次）才产生中断。输入信号的范围是满电源电压——即分别使用AV_{SS}和AV_{DD}作为转换电路（即作为S/H电路的参考电压）参考电压的上下限。于是，

CH0SA = 0x02	(Mux A = 通道2)
CH0NA = 0	(Mux A 以V_{REF-}为参考)
CH0SB = 0x0E	(Mux B = 通道14)
CH0NB = 1	(Mux B以AN1为参考)
ADCHS = 0x1E02	(CH0NB位于第12位，CH0SB位于<11:8>,

① 冷结补偿是一种消除任何不同金属结上与温度有关的电压波动的方法。本书将在后面的多通道温度检测应用中详细讨论该问题。

	CH0NA位于第4位，CH0SA位于<3:0>）
ALTS = 1	（使能轮流采样）
BUFM = 0	（不分割ADCBUF）
SMPI = 5	（每中断采样次数 = 6）
CSCNA = 0	（取消扫描输入）
VCFG = 0	（$ADV_{REFH} = AV_{DD}$，$ADV_{REFL} = AV_{SS}$）
ADCON2 = 0x0019	（VCFG位于<15:13>，CSCNA位于第13位，SMPI位于<5:2>，BUFM位于第1位，ALTS位于第0位）
ADCSSL = 0x0000	由于不扫描，因此取值无关紧要

例子2：通道扫描模式采样

假设要采样通道1～3、6、8、10和12，并且希望在每通道采样2次后才产生中断（即每次中断内完成采样14次），此外，待转换信号范围是V_{REF-}～V_{REF+}。于是，ADC模块配置如下：

CH0SA = 0	(Mux A = 0，尽管这是个“无关紧要”的值)
CH0NA = 0	(Mux A 以V_{REF-}为参考——“无关紧要”的值)
CH0SB = 0	(Mux B = 0，——“无关紧要”的值)
CH0NB = 0	(Mux B以V_{REF-}为参考——“无关紧要”的值)
ADCHS = 0x0000	(CH0NB位于第12位，CH0SB位于<11:8>，CH0NA位于第4位，CH0SA位于<3:0>——都是“无关紧要”的值)
ALTS = 0	(取消轮流采样)
BUFM = 0	(不分割ADCBUF)
SMPI = 13	(每中断采样次数 = 14)
CSCNA = 1	(使能通道扫描方式)
VCFG = 3	($ADV_{REFH} = V_{REF+}$，$ADV_{REFL} = V_{REF-}$)
ADCON2 = 0xC434	(VCFG位于<15:13>，CSCNA位于第10位，SMPI位于<5:2>，BUFM位于第1位，ALTS位于第0位)
ADCSSL = 0x154E	使能<12>、<10>、<8>、<6>、<3:1>位，以采样通道1～3、6、8、10及12

81

例子3：交错和通道扫描组合方式采样

下面假设要采样通道2、5和7，并且将这组通道与通道4轮流采样，每个中断内各通道采样4次（共计16次）。待转换信号范围是V_{REF-}～V_{REF+}，并且负输入端的S/H电路接V_{REF-}。为此，将ADC模块配置如下：

CH0SA = 0	(Mux A = 0，尽管这是个“无关紧要”的值)
CH0NA = 0	(Mux A 以V_{REF-}为参考)
CH0SB = 4	(Mux B = 0，——“无关紧要”的值)
CH0NB = 0	(Mux B以V_{REF-}为参考)
ADCHS = 0x0400	(CH0NB位于第12位，CH0SB位于<11:8>，CH0NA位于第4位，CH0SA位于<3:0>)
ALTS = 1	(使能轮流采样)

BUFM = 0 (不分割ADCBUF)

SMPI = 15 (每中断采样次数 = 16)

CSCNA = 1 (使能通道扫描方式)

VCFG = 3 ($ADV_{REFH} = V_{REF+}$，$ADV_{REFL} = V_{REF-}$)

ADCON2 = 0xC43D (VCFG位于<15:13>，CSCNA位于第10位，SMPI位于<5:2>，BUFM位于第1位，ALTS位于第0位)

ADCSSL = 0x00A4 使能<7>、<5>和<2>位，以采样通道2、5和7

概括起来说，我们已经根据应用的特殊要求配置了I/O引脚、选择了电压参考源并且确定了需要转换的ADC输入通道。下面还剩下两步：指定ADC转换时钟和选择转换触发器。 82

e. 指定ADC转换时钟

每个完整的数字化周期（单通道的采样和转换）需要1个ADC时钟周期（记作T_{AD}）来获取模拟信号，每转换1位也需要1个ADC时钟，最后还需要1个ADC时钟将数字化结果传至ADCBUFx并启动下一轮数字化。这就意味着一个10位的ADC完成1次单通道数字化需要13个T_{AD}，而12位ADC就需要15个T_{AD}。对于大多数dsPIC系列DSC，时钟源和周期是通过软件配置的，但是芯片的硬件要求在供电电压不低于4.5V时的T_{AD}至少达到333.3ns，而当供电电压低于4.5V时则至少达到666.7ns。如果无法满足上述时间要求，那么保持放大器没有足够的时间在不同通道间切换并充电，于是将导致ADC读数比实际值小很多。

T_{AD}可由T_{CY}（系统时钟周期）计算得出，它们的关系为：

$$T_{AD} = T_{CY} \times (0.5 \times (ADCS < 5:0 > + 1)) \tag{3-1}$$

其中：

T_{AD}是ADC时钟周期，

T_{CY}是系统时钟周期，

ADCS<5:0>是6位ADC时钟选择寄存器ADCS。

根据该公式，我们可以计算出V_{DD}大于等于4.5V时的最大采样速率：

$f_{smax} = 1 / (15 \times T_{ADmin}) = 1 / (15 \times 333.3\text{ns}) = 200\ 020$ sps

当ADC在V_{DD}小于4.5V的条件下工作时，f_{smax}只有上述值的一半，即100 010sps。

为了得到配置ADCS寄存器的正确值，我们可以根据上述公式很容易计算出ADCS。

$ADCS<5:0> = 2 \times (T_{AD}/T_{CY})-1$

例如，如果我们想采样所有16个通道，且每个通道的采样速率为10ksps（总的采样速率为16 × 10ksps，即160ksps），我们就需要160ksps × 15 = 2 400 000个ADC时钟周期/秒。这对应的T_{AD}为1/(2 400 000周期/秒) = 416.7 ns。假设dsPIC系列DSC运行在30MIPS的最大速度下，那么每个指令周期为T_{CY} = 1/30 MIPS = 33.33 ns。于是我们可以计算出ADCS的值为：

ADCS = 2 × (416.7ns/33.3ns) –1 = 24

特别需要注意的是，ADCS寄存器只有6位，这限制了它的最大值仅为63。因此，当dsPIC系列DSC运行在30MIPS速度下进行12位A/D转换时，T_{AD}的最大值为： 83

$T_{ADmax} = 33.33\text{ns} \times 0.5 \times (63+1) = 1.07\text{ns}$

它对应的最小采样频率为：

$f_{smin} = 1/(15 \times T_{ADmax}) = 1/(15 \times 1.07\ \mu\text{s}) = 62\ 507$ sps

如果应用程序所用的算法需要更慢的采样率，那么应用程序就必须要么抽取采样数据[1]，要么利用轮询和中断驱动的组合形式启动所有运行在更高采样率的通道完成一次数字化。此外，还有其他的方法，比如用软件定时器实现以期望的更低速率进行采样和数字化。

最后一种方法在它刚刚出现时效果也不错，尽管这会产生额外的软件开销，但是该方法使得我们能够相对同步地采样所有通道，这个功能对于那些通道间信号有关联的系统特别重要。比如，机器人手臂系统中经常遇到所谓的*臂端力矩传感器*，它被用于测量物体对机器人手臂末端产生的力和力矩。这使得机器人能够安全地移动物体而不会将它掉下（如果机器人的握力不足就可能掉下来）或者捏碎（如果该机器人使力太大极可能捏碎），并且通常会测量三维方向上的力和力矩。一种非常普通的力/力矩传感器，它能测量机器手内部三个精确定位的物理框架上的载荷，因此来自这三个框架的信号彼此相关。如果各通道所测信号的采样时间差太大，那么有些采样值是取自前一个时刻的信号状态，而有些采样值则是后一时刻的信号状态，系统就会因此而丢失信息。为了得到最精确的读数，就要尽可能同步地测量所有信号。通过减小信道间的采样时间，我们能够显著提高系统性能，因为来自同组的采样值变得“更加同步了”。

f. 选择采样和转换触发器

配置ADC模块的最后一步是选择采样和转换触发器。如前文所述，每个数字化周期包括了采样阶段和转换阶段。在采样阶段，采样/保持放大器在输入端为当前的模拟信号电压充电；在转换阶段A/D转换器电路将采样/保持放大器的模拟信号转换成数字值。在转换阶段的最后，数字化结果会被存储到合适的ADCBUF中。dsPIC系列DSC允许用户控制如何启动这两个阶段（即所谓的*采样触发器*和*转换触发器*），从而为设计师带来了很大的灵活性。

84

采样触发器有两个选项：一种是手动触发，即应用系统在需要转换输入信号时设定采样标志；另一种是自动触发，即在完成转换当前通道后会立即启动转换下一个通道。手动触发方式适用于检查非周期性的被监控信号（比如，在后台监控液位以确定它没有溢出）或者要求的采样速率低于自动转换模式支持的最小速率的情况。后一种情况的例子是采样热质量[2]大的器件的温度，此时由于物体温度变化相对较慢，应用系统为了减小软件开销因此很少采样它。尽管采样速率较慢，但是仍要求周期性采样，以便正确地实现DSP算法。为了缓慢且周期性地采样信号，应用程序可以在定时器的中断服务程序里，以期望的采样周期进行数字化。

应用程序可以通过配置ADCON1寄存器的A/D采样自动启动（ASAM）位（ADCON1<2>）来选择期望的采样触发器。清除ASAM位会将ADC模块配置成采用手动采样触发器，此时应用程序必须在每次需要采样输入信号的时候对ADCON1寄存器的A/D采样使能（SAMP）位（ADCON1<1>）置“1”。相反，如果将ASAM位置1，那么ADC模块则会自动采样，SAMP位会由硬件自动置位，而不用应用程序干涉。无论是手动还是自动采样，

① *抽取*是指将以一定速率采样的结果转换成采样速率更慢的结果。如果高速采样速率是低速采样速率的整数倍，那么只需简单地忽略高速采样结果流中采样序列号不是低速采样数据流序列号整数倍的结果即可。然而，如果高速采样速率不是低速的整数倍，那么必须先通过在高速采样结果中*插值*以得到第三个数据流，它的采样速率是原来高速和低速采样速率的最小公倍数。然后就可以按照前述方法对第三个数据流进行抽取，从而得到期望的低速率采样结果。

② *热质量*是与机械学的物理质量类似的热控学中的概念。物体的热质量越大，那么它的温度就越难改变。例如，大尺寸钢材的热质量就比较大，它需要很多的热量才能被加热，但是它一旦被加热，又可以长时间保温。而细针的热质量相对较小，它可以被迅速加热或冷却。

采样保持放大器都会采样输入信号，直到转换触发器启动转换阶段。

应用程序通过配置ADCON1寄存器的转换触发源选择（SSRC<2:0>）位（ACON1<7:5>）来决定启动转换阶段的方式。尽管3个SSRC位最多可以对应8种可能的转换触发源，但是其中有3种是保留的，见表3-1。请注意，一旦SSRC位被置“1”，那么除非芯片掉电或者重写ADCON1寄存器，否则其内容不会发生改变。

表3-1 转换触发源在特殊功能寄存器ADCON1中的位映射

SSRC<2:0>	转换触发源
000	清除SAMP位，以结束采样并启动转换
001	INT0管脚发生变化时结束采样并启动转换
010	发生定时器3的比较事件时结束采样并启动转换
011	电机控制PWM脉宽溢出时结束采样并启动转换
100	保留
101	保留
110	保留
111	由内部计数器结束采样并启动转换（自动转换）

85

采用手动转换时，应用程序只需将SSRC配置成000b并清除SAMP位就能结束采样阶段并启动转换阶段。应用程序等待转换结束时，既可以等待一段时间，也可以通过检查ADCON1寄存器中的A/D转换状态标志（DONE，ADCON1<0>）位来判断转换是否结束，当转换结束时，DONE位就会被置“1”。通过这样反复地触发SAMP位和检查DONE位，就能完成连续采样。

设计师还可以通过产生三种事件来启动转换阶段，这三种事件是：INT0信号（外部信号源）发生变化、dsPIC系列DSC的定时器3溢出（内部信号源）以及芯片的电机控制PWM脉宽结束（这也是内部信号源）。要想使用这些方式，应用程序必须将SSRC域配置成期望的触发源（001b、010b或者011b）并且正确配置对应的资源（外部中断、定时器或者PWM模块）。一旦发生了所选的事件，模块就会停止当前的信号采样并且开始转换阶段。

最后，应用程序还能使用*自动转换模式*，此时dsPIC系列DSC会在采样阶段开始后数个A/D时钟周期（1～32个周期）后自动启动转换阶段。在这种模式下，ADC模块使用内部计数器来监测采样阶段的时间，一旦到达指定时间（即T_{AD}）就立即启动转换阶段。将自动采样和自动转换模式相结合，应用程序就能使系统全自动运行，一旦启动就无需用户干涉。

2. 使用ADC模块的检查表

为了利用ADC模块完成一次A/D转换，应用程序必须执行以下7步基本操作。

（1）根据应用系统的喜欢通路要求配置ADC模块，包括：

- 将相关的I/O端口配置成模拟输入端口，选择合适的电压参考信号，以及将其他I/O端口配置成数字输入或输出端口；
- 选择需要进行转换的ADC输入通道；
- 选择期望的ADC转换时钟和触发器；
- 启动ADC。

86

（2）如果应用系统采样中断驱动式数据采样方式，那么按如下步骤配置ADC中断：

- 清除A/D中断标志位（ADIF）；
- 设置ADC中断的优先级。

（3）开始采样模拟信号。

（4）等待必要的数据获取时间。

（5）在数据获取结束时触发，并启动数据转换。

（6）以如下任何一种方式等待数据转换结束：

• 等待ADC中断；

• 监视ADC结束标志位，看它何时被置位。

（7）从ADC结果寄存器（ADCBUF0:ADCBUF15）中读取转换结果，如果执行中断驱动式获取，还要清除ADIF标志位。

3.3.2 定时/计数器模块

定时/技术器模块使dsPIC系列DSC能够产生精确的内部时钟，它们可以用于维持实时时钟（real-time clock，RTC），来追踪特定应用系统事件的占用时间，来调度时间优先事件（比如ADC采样触发器），或者计算某些外部硬件信号的过渡时间。根据所使用的dsPIC系列DSC的模型，模块可支持3个或者5个独立的定时/计算器，它们都能独立工作或者按照某种预定的组合方式产生大量灵活的时序和事件计数功能。

dsPIC的定时模块支持3种基本定时/计数器配置，这在文档里被记作A型、B型和C型定时/计数器。所有这三种类型都有些共同的特点，但是它们也有一些各自独有的特点，以便应对特殊的任务。在研究定时/计数模块时，我们将首先研究所有dsPIC的定时/计数器的共同点，然后再研究每种定时/计数器的独有特性，以了解设计师是为何以及如何在应用中采用某种配置。请注意，在下面的讨论中，我们将频繁使用寄存器或引脚名，比如PRx、TMRx、TxCON等。这里，只要用期望的定时器序号（1～5）代换“x”就能得到实际的寄存器或者引脚名称。比如，定时器1的寄存器有PR1、TMR1和T1CON，而定时器2的寄存器有PR2、TMR2和T2CON。

87

1. 通用定时/计数器的特性

每个定时/计数器都至少有如下特性：

（1）1个16位的周期寄存器（PRx）；

（2）1个16位的计数器（TMRx）；

（3）1个16位比较器，它能在每个时钟周期里比较计数器和周期寄存器的值；

（4）1个时钟预分频器，用于控制定时/计数器的更新频率；

（5）2个或者更多个定时/计数时钟源，其中之一是指令周期时钟（T_{CY}）；

（6）1个带有门控的定时器累加选项，它允许应用程序来测量定时器时钟输入引脚跳动的高还是跳动的低；

（7）能够设置中断标志并能在比较器检查到计数值与存储在周期寄存器中的值相等时产生中断；

（8）1个定时器配置寄存器（TxCON），使应用程序配置某个定时器的操作。

定时器的基本操作过程非常简单，应用程序先对定时器的计数寄存器（TMRx）清零，然后向定时器的周期寄存器（PRx）内写入数据，最后置位定时器控制寄存器（TxCON）里的定时器开关（TON）标志位，以启动定时器。定时器一旦被启动，它就会在所选时钟信号的每个上升沿将计数寄存器的数值加1。当计数寄存器的值与周期寄存器的值相等时，定时器就会置位中断标志位，并且产生中断信号（假设该中断已被启用）。置位中断标志时还会

将计数寄存器清零，且不用软件干预。上述过程会反复执行，直到用户清除TON标志，才能关闭定时器。

尽管在上述情况下定时器总能正确工作，但是在总结基本定时器的工作方式前，还需要读者掌握几个关键概念，以便能够正确配置定时器。第一个是如何为定时/计数器选择合适的时钟源，第二个是时钟预分频器的配置，第三个是如何计算需要载入周期寄存器的值，它由所选的时钟源和时钟预分频器决定。下面我们就将重点研究这三个问题。 88

应用程序可以配置定时器使用基于指令周期时钟的内部时钟（其周期为T_{CY}），或者TxCK引脚上的外部时钟信号（其周期为T_{XCK}）。无论是哪种情况，所选的时钟源都会经过预分频器，并先被分频系数1、8、64或者256分频（即时钟周期被分别乘以1、8、64或256），然后才被相关的计数寄存器使用。这就使芯片能够实现更长时间的定时周期，但代价是时间分辨率下降，这一点我们很快就可以从下面的例子中看出。如果选用运行速度为30MIPS（对应的T_{CY}=33.33ns）的dsPIC系列DSC的内部指令周期时钟作为时钟源，且预分频系数（PSF）等于1，那么所得的定时器分辨率为：

$$T_{RES} = T_{CY} \times PSF = 33.33\text{ns} \times 1 = 33.33\text{ns}$$

而16位计数寄存器将支持的最大周期值为：

$$T_{MAX} = 65\,536 \times 33.33\text{ns} \approx 2.18\text{ms}$$

另一方面，如果使用相同的时钟源，但是所用的预分配系数等于256，那么定时器的分辨率会下降为：

$$T_{RES} = T_{CY} \times PSF = 33.33\text{ns} \times 256 = 8.5325\ \mu\text{s}$$

但是16位计数寄存器现在支持的最大周期值为：

$$T_{MAX} = 65\,536 \times 8.5325\mu\text{s} \approx 559.18\text{ms}$$

一旦选定时钟源和预分频系数，就可以很容易地计算出载入周期寄存器的数值：

$$PR = T_{PERIOD} / T_{RES}$$

其中，

PR是载入周期寄存器（PRx）的16位数据；

T_{PERIOD}是期望的周期，单位为秒；

T_{RES}是定时器的分辨率，单位为秒。

当然，T_{PERIOD}和T_{RES}必须确保周期寄存器的值符合16位寄存器字长的要求——即PR必须小于或等于65 535。在很多应用中，都要求满足寄存器字长的要求，它将决定时钟源和预分频系数的选择。

至此，我们已经看到所有三种定时/计数器的共有特征，下面就研究一下它们的区别，以及如何针对不同场合选用类型合适的定时/计数器。

2. A型定时/计数器

A型定时/计数器可用作16位定时器（如前所述）、16位同步计数器或者16位异步计数器。 89 当它用作16位同步计数器时，该定时/计数器的行为和用作16位定时器时基本相同。二者唯一的区别是，在定时器模式下，我们希望时钟稳定且具有周期性；在计数器模式下，则希望它能对非周期出现的输入信号（通常由相关的外部时钟引脚输入）计数。定时器的硬件能够在将同步时钟送往计数器之前将计数信号与内部相位时钟同步，在外部时钟信号的每个上升沿都能使计数器加1。在定时器模式下，当TMR1寄存器的值等于周期寄存器PR1的值时，TRM1寄存器被清零，并产生中断。

异步计数器模式是A型定时/计算器独有的工作模式，并且能使用32kHz低功率振荡器作为外部时钟信号。这两个特性使得它可以在dsPIC应用中作为理想的实时时钟（RTC）。A型定时器在系统时钟无效时也能工作，也就是说当芯片在休眠态时定时器仍在工作，这是真正的RTC必须具备的能力。在任何dsPIC系列DSC中，只有一个A型定时器——定时器1。图3-12是A型定时/计数器的框图。

图3-12　A型定时/计数器框图

在图3-12中有几个要点需要注意。首先，由于定时器1在其时钟输入（T1CLKI）处具有特殊电路和辅助振荡器输入（SOSCI）信号，因此，应用系统可以使用一个外部低功耗的32kHz晶体为该定时器产生时钟。第二，无论是否使用时钟源，时钟信号都将在与内部指令周期时钟同步前通过预分频器，从而使输入时钟的频率至少可以达到$0.5 \times (F_{CY} \times PSF)$（即输入时钟可以比指令周期时钟快$1/2 \times PSF$倍）。但是，由于受到用于实现异步外置时钟输入的组合逻辑电路的时间约束，通向A型定时/计数器的外部时钟信号的频率必须小于25MHz才能正常运行，因此这限制了预分频系数期望的频率范围。当然，和所有dsPIC的定时/计数器一样，外部时钟必须满足上升和下降时间的要求，最大限制为10ns。

90

既然知道了A型定时/计算器的用途，那么需要执行哪些步骤才能配置它、使它正常工作呢？首先，我们必须确定希望的定时/计数器工作方式，定时器、同步计数器或者异步计数器（该方式用于实现实时时钟）。确定了工作方式后，我们需要决定在何时让定时/计数器产生中断。（即计数器达到何值时定时/计数器会向芯片发出中断。）

a. 定时器模式初始化

为了初始化定时器1，使之工作在定时器模式，必须完成以下工作。

（1）在初始化时，先清除T1CON寄存器的定时器启动（TON）控制位（T1CON<15>），以停止定时器。然后清除IFS0寄存器中中断标志位（T1IF）以及IEC0寄存器中的定时器1中断使能位（T1IE），以清除所有未决定的定时器1的中断条件。这样能保证初始化过程不受中

断影响地按顺序完成。

(2) 清除T1CON中的定时器时钟选择（TCS）标志位（T1CON<1>），以选择内部指令周期时钟作为定时器预分频器的输入。

(3) 通过配置T1CON中的定时器时钟预分频（TCKPS）位（T1CON<5:4>）将时钟预分配器设为期望值：

TCKPS = 00b　　使用1:1时钟预分频系数（即无预分频）
TCKPS = 01b　　使用1:8时钟预分频系数
TCKPS = 10b　　使用1:64时钟预分频系数
TCKPS = 11b　　使用1:256时钟预分频系数

(4) 由于使用内部时钟源，因此无需同步外部时钟信号，故应清除T1CON中的定时器外部时钟输入同步选择标志（TSYNC）位（T1CON<2>）。

(5) 根据应用程序要求定时器在空闲模式停止或运行，配置T1CON中的定时器在空闲时停止标志（TSIDL）位（T1CON<13>）。

91

TSIDL = 0　　允许定时器在芯片进入空闲模式后运行
TSIDL = 1　　禁止定时器在芯片进入空闲模式后运行

(6) 根据应用程序是否使用门控时间累计模式，配置T1CON中的定时器门控时间累计使能标志（TGATE）位（T1CON<6>）：

TGATE = 0　　禁止门控时间累计模式
TGATE = 1　　允许门控时间累计模式（此时必须使用内部时钟源，因此当TGATE=1时，TCS位必须被清零）

(7) 根据希望的中断间的占用时间来配置周期寄存器（PR1），它的计算方法已在前面的第1小节介绍过。

(8) 如果应用程序需要在定时器溢出时产生中断（即当TMR1的值等于第(7)步所设定的PR1的值时，或者在门控时间累计模式使能条件下门控信号的下跳沿时刻），那么要将IEC0寄存器中的定时器1中断使能位（T1IE）配置成允许中断。此外还要使能全局中断（如果还未使能的话）。

(9) 置位T1CON寄存器的TON位，启动定时器。

请注意，在初始化过程中，第(2)~(6)步通常合在一起，一同写入T1CON寄存器，这里将它们分开只是为了描述得更清晰。

b. 同步计数器模式初始化

同步计数器模式与定时器模式类似，只是这里被计数的信号是外部时钟输入信号，并且需要由内部时钟对其同步。在该模式下，不使用门控时间累计的概念。因此，其初始化步骤与定时器模式非常类似。

(1) 与定时器模式初始化部分的第(1)步相同。

(2) 置位T1CON寄存器的定时器时钟选择标志位，以选择外部时钟输入。

(3) 与定时器模式初始化部分的第(3)步相同。

(4) 置位T1CON寄存器的定时器外部时钟输入同步选择标志位(TSYNC)，以便将外部时钟输入同步到指令周期时钟上。

92

(5) 与定时器模式初始化部分的第(5)步相同。

(6) 清除T1CON寄存器中的定时器门控时间累计使能位(TGATE)，因为我们不需要使用

门控时间累计模式。

(7) 与定时器模式初始化部分的第(7)步相同。

(8) 与定时器模式初始化部分的第(8)步相同。

(9) 与定时器模式初始化部分的第(9)步相同。

和定时器模式的初始化过程一样，第(2)～(6)步通常是通过对T1CON寄存器的一次写操作一同完成的。

c. 异步计数器模式

异步计数器模式与同步计数器模式基本一样，区别在于这里的计数器可以在芯片进入休眠模式后由一个低功耗的32kHz晶体振荡器配合工作。如果采用晶体振荡器作为时钟输入，那么振荡器的一侧与外部时钟输入信号端口相连，而另一侧则与辅助振荡器输入引脚（SOSCI）相连。此外，振荡器控制寄存器（OSCCON）中的低功耗振荡器启用标志位（LPOSCEN）也必须被置为“1”，以启动驱动该晶体所需的晶体振荡器支持电路。

正如我们所期望的那样，它的初始化过程与同步计数器的非常类似。

(1)～(3)与同步计数器初始化部分的第(1)～(3)步相同。

(4)清除T1CON寄存器中的定时器外部时钟输入同步选择标志位（TSYNC，TICON〈2〉），以便外部时钟源工作在异步方式。

(5)～(9)与同步计数器初始化部分的第(5)～(9)步相同。

和其他初始化一样，其中第(2)～(6)步通常被集成在对T1CON寄存器的一次写操作内完成。

3. B型定时/计数器

和A型定时/计数器一样，B型定时/计数器可以用作16位定时器或者16位同步计数器。但是，它们还能和C型定时/计数器组成32位定时器或32位同步计数器。当组合成这种方式时，B型定时/计数器作为32位的低16位，而C型定时/计数器则作为高16位。这样能够增加灵活性，并且代价很小。尽管B型定时/计数器无法工作在异步模式，也无法接受晶体振荡器输入，但幸运的是，这个问题在大多数应用中并不严重。

93

在支持B型定时/计数器的器件中，定时器2和定时器4（如果存在的话）就属于这种类型。此外，当将B型和C型定时/计数器配对时，只能工作于几种特定的组合方式。定时器2只能与定时器3组合，而定时器4则只能与定时器5组合。当组合成一个32位单元后，B型定时/计数器的定时器控制寄存器（TxCON）将作为整个32位定时/计数器的控制寄存器。由于dsPIC系列DSC无法直接读/写由TMR3/TMR2或者TMR5/TMR4组成的32位数据，因此，芯片使用保留寄存器（TMRxHLD）存放这些定时/计数器的高位字。为了向组合的定时/计数寄存器写入32位数据，应用程序要首先将高16位写入TMRxHLD寄存器，然后再将低16位数据写入B型定时/计数器的TMRx。当应用程序向TMRx写入数据后，dsPIC系列DSC会立即将TMRxHLD的值载入对应的C型定时/计数器的TMRx寄存器。从组合型计数寄存器中读取32位数据的过程与此相反，当应用程序从B型计数寄存器读取低位字时，dsPIC系列DSC会自动将高位字的数据载入TMRxHLD寄存器，然后应用程序就能从TMRxHLD寄存器中得到高位字的数据。

图3-13是B型定时/计数器的框图。它与A型定时/计数器的唯一显著区别是B型定时/计数器的框图中没有用于支持晶体振荡器输入的信号调理电路以及从预分频输出到计数电路的异步时钟通道。和A型定时/计数器一样，B型定时/计数器的预分频器位于同步电路之前，因此

B型定时/计数器能够接受频率达$0.5 \times F_{CY} \times PSF$的外部时钟信号。由于外部时钟信号输入部分没有异步组合逻辑，因此B型定时/计数器的时钟信号不受像A型电路的25MHz最大频率的限制。

图3-13 B型定时/计数器的框图

B型定时/计数器在TxCON寄存器中还有A型和C型定时/计数器都没有的标志位。应用程序通道置位定时器模式选择位（T32）后，就能将B型定时/计数器和对应的C型定时/计数器组合起来，并能自动清除允许这两个定时/计数器以单独的16位方式工作的标志位。由于B型定时/计数器总是工作在同步方式，因此它们的TxCON寄存器不需要TSYNC位，故而也没有定义该标志位。

a. 16位定时器和16位同步计数器初始化

B型定时/计数器的初始化过程和A型定时/计数器用作16位定时器或者16位异步计数器时的初始化过程基本相同。只是在初始化时必须将TxCON中的T32位清零，并且不需要初始化TSYNC位（由于它根本不存在于B型单元内）。

b. 32位定时器或同步计数器初始化

B型定时/计数器工作在32位工作方式时，除了必须置位TxCON的T32位，而且必须使用B型定时/计数器的中断使能标志外，其他与用作16位定时器的初始化过程基本相同。注意还要配置B型和C定时/计数器的PR寄存器。

4. C型定时/计数器

C型定时/计数器与B型定时/计数器类似，并且具有相同的工作方式。但是，如图3-14所示，其外部时钟同步电路位于预分配器之前而不是之后，因此C型定时/计数器的最大外部时钟频率仅为$0.5 \times F_{CY}$，而达不到B型定时器的$0.5 \times F_{CY} \times PFS$那么高。和B型定时/计数器一样，由于外部时钟输入通路上没有异步组合逻辑，因此它的时钟频率也不受像A型单元的25MHz

那样的限制。

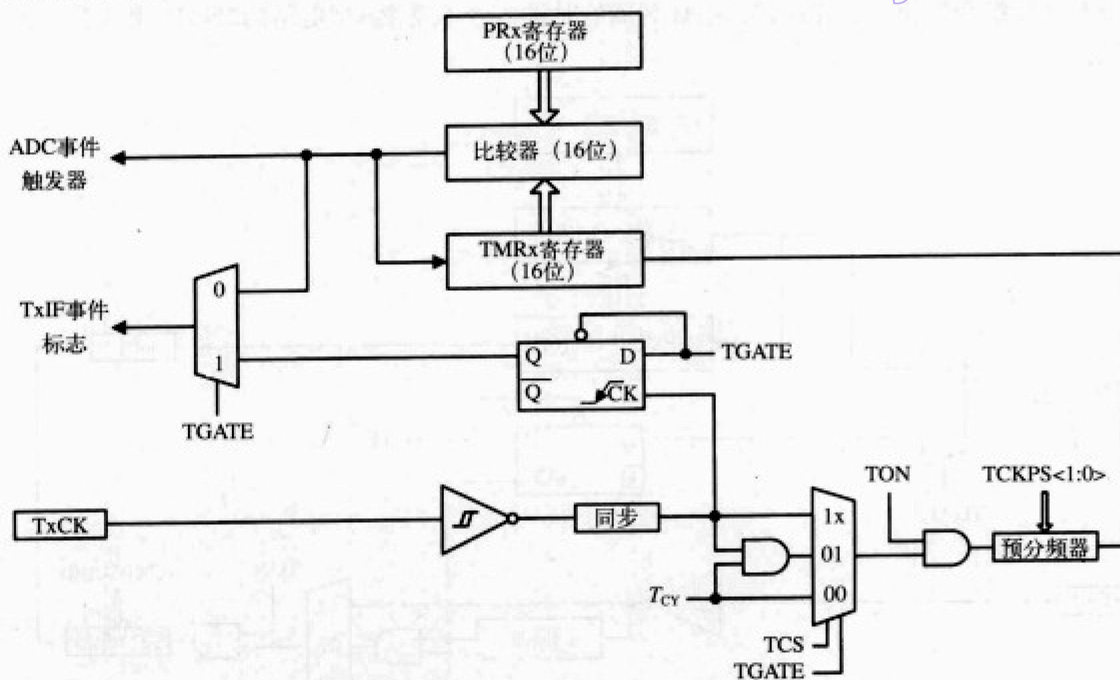


图3-14 C型定时/计数器的框图

C型定时/计数器的初始化过程与B型定时/计数器类似，它们都有四种不同的工作方式。但是，其中有一个重要的区别。当C型定时/计数器工作在16位模式时，它与B型定时/计数器的控制寄存器对应的T32标志必须被清零，这样才能使这两个定时/计数器都工作在16位模式。

95

3.4 小结

本章详细讲述了dsPIC系列DSC的架构，当然，不可能涉及该芯片的所有方面。有关该系列芯片其他部分的详细介绍可参阅Microchip公司的技术文档，比如*dsPIC30F Programmer's Reference Manual*、*dsPIC30F Family Reference Manual*以及应用系统所采用的dsPIC器件对应的数据手册。所有这些资料都可以从Microchip公司的网站（www.microchip.com）上获得，该网站还有大量的应用笔记、设计提示以及介绍和研究某种系统器件及其实际应用问题的网络研讨班资料等。

下一章，我们将通过挖掘该芯片的通信资源而继续研究dsPIC系列DSC具备的外围设备模块。

96

第4章　智能传感器通信的实现

我们在研制智能传感器时会遇到的一个贯穿始终的问题，就是智能传感器与本地或者远程的其他部件的通信能力如何，这也是衡量智能传感器性能的一项重要指标。正如我们所看到的，dsPIC系列DSC提供了多种通信接口，设计者可以根据传输数据的特征、传输速度以及传输的物理媒介等因素选择最适合的通信接口。本章内容可分为两部分，前一部分介绍在智能传感器应用中常见的各种通信要求，后一部分则介绍dsPIC系列器件提供的通信接口，指出每种接口最适合的应用环境，并详细介绍如何使用相关的硬件模块。

4.1　通信的类型

既然要研究通信问题，那么就必须理解，智能传感器的任何通信形式都没有人类的通信方式好。当我们希望向某人传递信息时，就会本能地来调整音量、语速以及传递的信息量，这些可根据当时的环境、谈话内容的重要程度以及谈话对象来决定。假如是带着四岁的侄女逛卢浮宫，那么最常用的词语就是："多么美丽的图画啊！"但如果是陪一名专业的画家，那么就可能在讨论笔画、色彩以及风格。在这两种情况下，由于听众不同，因此谈话的方式也不同，信息量和内容的深度也不同。

这种情况对智能传感器来说没什么不同，我们所采用的通信手段取决于希望传输的数据类型、紧迫程度以及从信息源到目的地的传输媒介等。例如，配置命令通常是在传感器开始报告参数的测量值前，由主机发往传感器系统的。除非命令数据的数量非常大，否则其传输速率就不需要很高，关键问题是指令必须按照发送的顺序依次到达传感器。这种通信并不是完全由时间来确定其优先级的。相反，假如传感器检测到某个报警条件，那就必须可靠而迅速地将其传输至主机，以便根据该信息避免不想要的甚至是危险的工作条件。显然，我们希望处理后一种情况的通信技术比前一种更可靠并且更迅速。幸运的是，dsPIC系列DSC提供了很多选择，能满足各种通信情况的需要。

4.1.1　通信信道的重要特性

尽管前面已经接触到一些有关通信信道的重要特性，但是这里的讲述将更加明确，以便我们能够更好地区分不同的方法，并提供一些基准，以便能据此选择最合适的通信条件。在下面的讨论中，都假设采用有线电子通信信道（而不是光、无线或者其他媒介）。读者可能认为这一假设是"理所当然"的，但是请注意，有很多系统是采用光或者无线通信的。这里之所以要做这样的简单假设，目的是以此为例来介绍dsPIC系列DSC的通信接口。

在构建通信系统时使用到的主要准则有以下几条。

（1）信道的持续（与此相反的是迸发）数据吞吐量要求，单位是bit/s。

（2）通信链路采用点到点方式（即链路上只有两台设备）还是多点方式（同一条链路上有2台以上的设备）。

（3）通信链路的物理长度和电气噪声等级。

（4）通信链路是同步的还是异步的，还是两种都有。

（5）硬件错误检测要求。

在展开介绍dsPIC系列DSC的通信条件前，让我们首先了解一下上述准则。

4.1.2 信道的数据吞吐量

信道的*数据吞吐量*是最为重要的通信信道特征之一，它是指单位时间内能够传输的数据量。数据吞吐量有两种基本类型：*迸发吞吐量*和*持续吞吐量*。设计师必须理解这两个名词的区别以及在某个应用中哪个是约束性因素。迸发吞吐量是指信道支持的传输少量数据的最大传输速率。只掌握迸发吞吐量还不够，设计师还要掌握在迸发吞吐量下的最大传输数据量。尽管信道的迸发吞吐量可能很高，但是如果它无法在规定的情况下传输足够多的数据，那么也不能满足要求。持续吞吐量是指信道连续传输数据的最大速率。由于持续吞吐量反映的是与传输数据量无关的传输速度，因此它可能比迸发吞吐量小很多，但是它能更好地评估信道的数据承载能力。

在我们设计传感器时，主要关心的是信道的持续数据吞吐量，当然，我们也会指出，在某些情况下用迸发模式可能更合适。

4.1.3 点对点和多节点网络

通信系统可分为两类。一类是采用点到点的拓扑系统，它只有两个节点可以相互传输数据，具体参见图4-1。尽管可以利用在线的中继器接收某个节点的数据再传送至另一个节点，但是关键问题是每个链路中一个方向上只能有一对接收器/发送器。由于只有一个发送器，因此它不必关心数据传输的时刻表。这使得设计师的工作相当简单。此外，采用点到点拓扑还能显著简化数据线缆的终端器，这是长线传输中必需的。

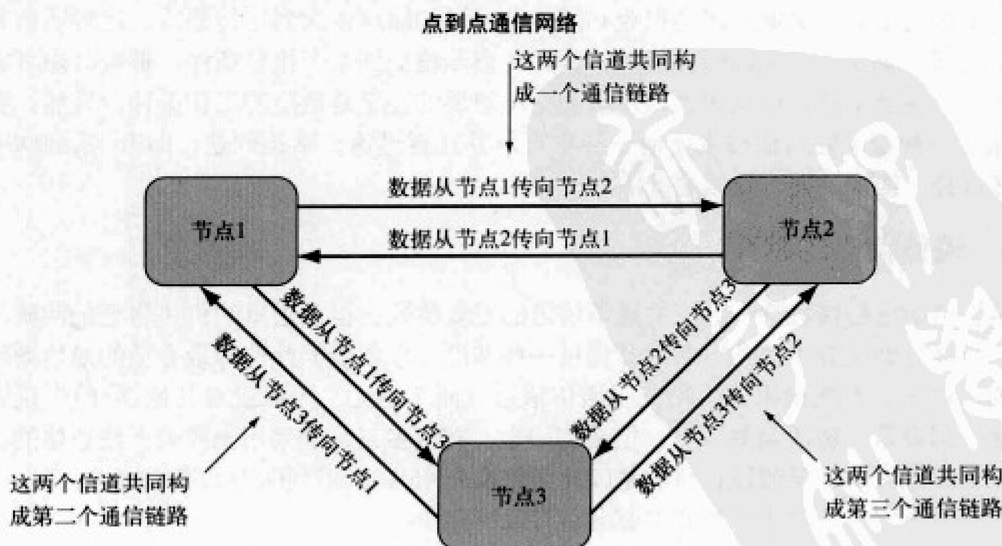


图4-1 点到点通信网络

点到点型系统的主要缺点是它与其他系统共享数据的能力有限。这就好比当你正和某人谈话时很难和另一个人插话，因此很多系统都采用*多点*拓扑，它能在单个通信信道上连接两

个以上的节点，具体参见图4-2。利用这种方法，就能够从一个节点同时向所有节点发送信息，或者实现两个或更多个节点相互通信。注意，尽管允许让多个节点同时监听，但是同一时刻只允许有一个节点发送数据，由此导致这种拓扑的最大挑战是：如何安排各个节点的发送顺序，以便两个或者更多个节点不会同时发送数据。 101

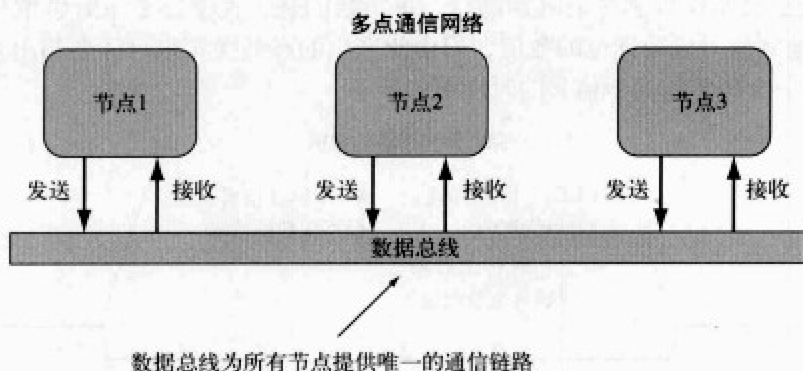


图4-2 多点通信网络实例

通常有两种方法可以克服传输冲突（冲突是指某一时刻有不只一个发送器要发送数据的情况）。第一种方法是，通过传递一个软件或者硬件令牌，告知每个发送器何时可以发送数据安全。当发送器得到令牌后，它就能发送任何数据，待发送完毕后再将令牌传递给网络中的其他节点。如果接收令牌的节点没有数据传输，那么它只是将令牌传递给下一个节点，而不做其他任何事情。尽管这种方法可以避免冲突，但代价是在网络中顺序地传递令牌增加了软件或硬件开销，并且会花费（相对）较多的时间将令牌在无数据可发的节点间传递，而需要发送数据的节点又得等令牌。 102

第二种方法采用了不同的技术。它不是避免冲突，而是允许发生冲突，但是可以迅速而可靠地检测到冲突，然后重新安排，让其中至少一个发送器在主发送器（即将发送数据的发送器）发送完毕后再发送。该方法能够显著提高那些节点是非周期性地发送数据的网络的吞吐量，原因是它允许节点立即尝试发送，而不必等着轮到自己才发送。当然，第二种方法中的重排序算法必须确保所有节点都能得到发送权，以防止某个节点长久无法发送数据。

4.1.4 数据链路的物理属性

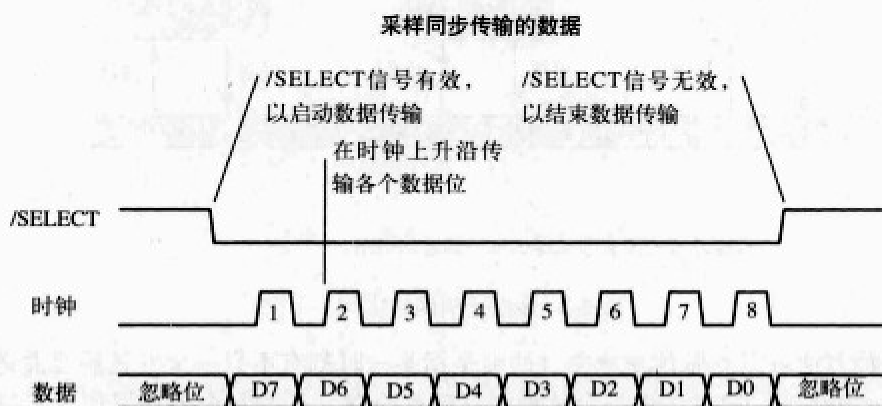
数据链路的物理属性包括链路长度和工作环境的电气噪声等级等，它们限制了最大传输速率，并可能导致必须使用特殊的电路来充分放大电信号，或者检测错误条件，或者两种情况都要处理。然而，使用高驱动能力的差分信号就能在一定程度上克服上述问题，但是增加的功耗和成本可能会超出应用的承受能力。正如我们想象的那样，为了实现可靠接收，链路越长或者环境噪声越严重，传输速率就越慢。

4.1.5 异步和同步数据传输

通常，既可以按照*同步方式*（有一个辅助时钟信号），也可以按照*异步方式*（没有辅助时钟信号）传输数据。每种方式各有利弊，应当根据应用的具体情况来选择合适的方式。

同步数据传输方式的优点是，发送器会产生一路时钟信号，它能精确地告诉接收器何时

采样传来的数据，以确定该数据是逻辑“0”或“1”（这里假设处理的是二进制数据）。这可 103 以极大地简化接收端的精确数据检测器的设计，并能保证始终以正确的时序进行通信，但缺点是成本较高。此外，大多数数据同步通信不但要求提供额外的一路时钟信号（如果时钟是以差分形式发送则是两路），而且还必须保证时钟信号非常干净，因为时钟信号上的任何“毛刺”都可能会被错误地解释成时钟翻转。更糟糕的是，通常还要求时钟信号具有陡峭的边沿以便清晰地判读对应数据位的数值，而由此产生的高频噪声很可能会超出系统允许的限制值。图4-3是一个简单的同步信号的时序图的例子。



要点

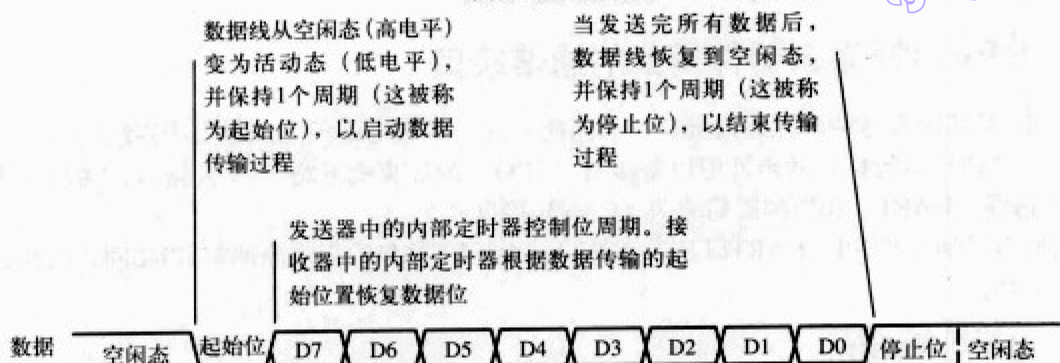
(1) 时钟信号能同步数据传输过程，以保证发送器和接收器采用同一个时钟信号传输数据

(2) /SELECT信号用于限定传输过程，即指示合法的数据传输过程。如果没有传输限制标志，时钟信号上的“毛刺”就可能启动一次错误的数据传输

(3) 为了方便图示，图中时钟的极性和相位可能与实际情况不同。而且实际的信号时序可能比这里显示的更复杂。例如，很多同步数据传输器都采用独立的数据接收和数据发送信号，以便能同时双向传输数据（即全双工传输）

图4-3　采样同步信号的原理

上面说了同步传输存在这么多问题，或许有人会断定异步传输是更好的方法，当然，有时这么说也是对的。异步链路的最大优点是它无需外部时钟信号，解码接收到的数据位本身就含有所需的时钟信息，从而完全消除了与外部时钟以及用导线传输该时钟信号有关的令 104 人头疼的问题。现在，读者应当很清楚这些优点是要付出成本的，这是毫无例外的。为了准确地恢复嵌入在数据流中的时钟，就需要在接收器中用高速时钟（通常是数据传输速率的4～16倍）同步接收到的数据流，这需要特殊的硬件并且限制传输速率的上限小于（通常是远远小于）同步系统能达到的最高速率。图4-4展示了异步信号时序的情况。



要点

(1) 发送器和接收器都包含了精确匹配的内部时钟，它们在起始位的下降沿被同步。如果这两个时钟的频率差别太大，那么接收器就会在错误的时间采样数据线，从而会报告错误的数据

(2) 要求数据线产生所谓的起始态和停止态（这里被标记为起始位和停止位）可以进一步增强数据流的同步性。任何不符合上述最小要求的传输都被视为是无效的

(3) 在数据流中增加额外的数据位还能进一步提供错误检测及纠错功能。这些附加位的取值由本次传输的数据决定。如果接收器发现读到的附加位数据与期望值不同，就会指出本次传输无效。当这些附加位被用于指示错误时，通常被称为校验位

(4) 为便于图示，本例中的某些地方，包括数据线用于指示特殊状态的电平以及数据位的顺序，可能与实际情况略有不同。此外，实际的信号时序也可能比这里展示的更复杂。例如，很多异步数据传输器都有独立的数据接收和数据发送信号，以便能同时双向传输数据（即全双工传输）

图4-4 采样异步信号的原理

105

4.1.6 硬件错误监测

在任何通信系统中，无论如何预防，总会发生错误。在这种情况下，设计师有责任构建一种能够自纠错的系统（至少也能够检测出错误），以便错误的数据不会导致执行异常的系统动作。根据数据链路的速度要求，也可以用软件实现这种错误检测，但是这会增加处理开销。

在有些情况下，根据链路速率或可靠性要求可能必须采用硬件错误检测，以保证收发器之间能够正常通信。遗憾的是，这种硬件解决方案会增加电路复杂性和成本，除非能将它们集成到标准通信设备中，并通过大批量生产来分摊相关电路的开发和生产成本。我们马上就会看到，dsPIC系列DSC至少含有一个通信接口——控制器局域网（CAN），它就符合上述要

求并且其硬件错误检测电路能保证通信链路高度可靠。

4.2 dsPIC30F系列器件具备的通信接口

dsPIC系列DSC提供多种通信接口，从而能与同一印制电路板上或者远程系统的其他设备通信。这些接口包括：同步外围设备接口（SPI）、内部集成电路（I^2C）接口、通用异步接收/发送器（UART）和控制器局域网（CAN）接口。

我们会详细介绍SPI、UART以及CAN接口。由于I^2C接口的很多原理与SPI相同，因此这里不做介绍。

4.2.1 串行通信接口

串行外围设备接口（通常称为SPI）是一种同步串行接口，主要用于同一块印制电路板（PCB）上的器件间传输数据，当然它也可以用于不同PCB之间的通信。该接口极为简单，包括一个串行数据输出信号（SDO）、一个串行数据输入信号（SDI）、一个串行时钟信号（SCK）、一个片选信号（CS）以及一个从选信号（SS）。这些信号都是单极性信号，这也是SPI总线不适合长距离或者在噪声环境中传输数据的原因之一。由于该接口很容易实现和维护，因此包括微处理器和外围设备芯片在内的很多器件都具有SPI。例如dsPICDEM开发板就有两个SPI，其中一个用于和板上的温度传感器通信，负责向传感器发送配置数据并且读取温度和状态值。

106

在过去几年里，SPI逐步发展到支持四种工作方式（分别称为方式1、方式2、方式3和方式4），这四种工作方式基本相同，只是SCK时钟沿和SDO及SDI数据信号的时序关系有所不同，从而决定何时发送数据、何时可以接收到有效数据。大多数器件都支持其中的某些方式，因此最重要的是确保发送和接收设备至少支持一种相同的工作方式。图4-5是四种可能的SPI工作方式组合。在后面的例子中，我们将采用方式0，因为它最为常用。

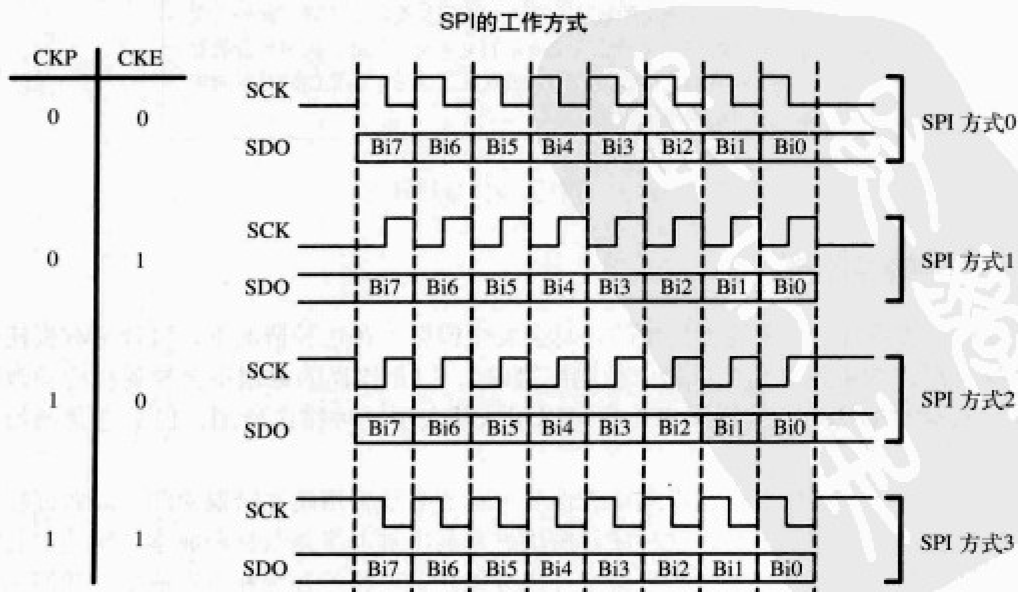


图4-5 SPI的四种工作方式

利用Microchip公司的16-bit Peripheral Library（16位外围设备函数库）就能控制并访问dsPIC系列DSC的SPI的程序框架。与后面将介绍的UART及CAN总线不同的是，SPI通常用于传输少量（通常是字节或者字）数据块。由于其传输速率很高，因此我们必须将传输过程视为在线实时进行的，就像传输的是要执行的代码一样。例如，如果我们正从一个通过SPI相连的传感器读取2B的温度数据，那么由于数据传输过程并不会明显减缓处理过程，因此我们还能处理传输时的请求和应答过程。但是当通过其他通信端口（或者即使是SPI）传输大量数据时，情况就有所不同。通常，我们得设计一个带有缓冲区的中断驱动的框架以应对这种情况。幸运的是，Microchip的16-bit Peripheral Library已经包含了我们使用SPI时所需的所有功能。 107

为了使用SPI，我们必须首先配置相关的dsPIC模块，包括配置数据传输速率、时钟极性、时钟相位（后两个决定了工作方式）以及中断极性等。

4.2.2 通用异步收发器

SPI和I^2C接口都是同步接口，而通用异步收发器（UART）则采用的是异步连接。UART可能是使用最广泛的通信接口，至少是电脑间最常用的通信接口，比如可靠的RS-232串行端口，直到现在它仍是个人电脑的标准配置端口。此外，RS-232串行端口还被广泛用于工业系统，并且是最早被广泛应用的标准接口之一。与RS-232类似的还有RS-422接口和RS-485接口，这是两个完全不同的UART接口。事实上，UART中的U就代表了这种通信方式的“通用性”。

基于UART的设备可以用于点对点拓扑中，也可以用于多点拓扑，但此时需要额外的驱动电路来关闭每个节点的发送器。UART通信中至少需要以下信号：一个接收数据（RxD）信号、一个发送数据（TxD）信号以及一个用作上述两个信号参考的地信号。实际中，某个节点的RxD信号与另一个节点的TxD信号相连，反之亦然。正如我们讨论过的，异步连接在发送的数据中嵌有时钟信号，或者至少有些办法可以使接收器用高速内部时钟来同步输入的数据流，以便提取出正确的数据位的值。对于一个基于UART的接口，这种同步是通过在每个数据字节的开始处插入起始位来实现的。数据依次排序，低位（LSB）在前、高位（MSB）在后。在数据之后是可选的校验位，以及一个长度为1、1.5或者2个位周期的停止位。图4-6展示了一个样本数据的发送过程。 108

请注意，空闲时RxD线为高电平，而起始位则是将该信号线从空闲态拉低到低电平（地），以通知接收器数据即将到来。这将导致UART启动内部位计时时钟以及控制采样时刻的硬件状态机。显然，链路两端应当配置成相同的传输速率，否则接收器的采样电路就可能在错误的时刻读取某个数据位，从而接收到无效的数据。 109

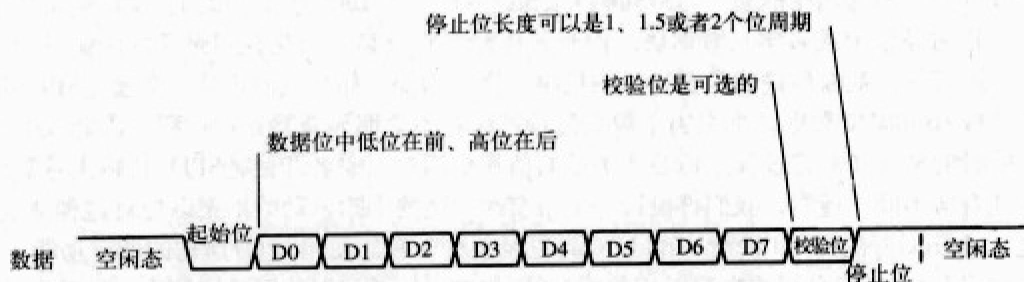
可选的校验位可用于基本的传输错误检测，尽管很多应用都忽略了它。当使用它时，校验位的值由发送器配置，而接收器则用该位检测接收到的对应数据。校验位有三种配置方式。

（1）无校验——发送器不产生校验位数据，并且不产生这个位周期。

（2）偶校验——当数据本身含有偶数个“1”时，该位被置“1”；当数据本身含有奇数个“1”时，该位被清零。

（3）奇校验——当数据本身含有奇数个“1”时，该位被置“1”；当数据本身含有偶数个“1”时，该位被清零。

UART数据传输过程的例子



要点

(1) UART数据传输的特征包括

(a) 比特率——每秒传输的位数

(b) 奇偶校验——产生奇偶校验码的方式

(c) 数据长度——每次传输的数据量

(d) 停止位的个数

(2) 比特率，有时也被误称作波特率，它决定了每个数据位的宽度并且表征了每秒传输的位数。常见的比特率有9600、19.2Kbit/s、38.4Kbit/s、57.6Kbit/s以及115.2Kbit/s。尽管并不一定要使用上述的传输速率（只要收发双方使用的传输速率相同即可），但是如果不使用这些标准比特率，就可能导致与其他系统通信失败

(3) 奇偶校验共有三种类型：偶校验、奇校验和无校验。当数据本身（不包括附加位，例如起始和终止位）包含有偶数个“1”时，偶校验会将校验位置“1”。而当数据本书包含有奇数个“1”时，偶校验就会将校验位清“0”

当数据包含奇数个“1”时，奇校验会将校验位置“1”，当数据本身包含偶数个“1”时，奇校验会将校验位清零

如果选择无校验，那么数据流中就不包含校验位

(4) 数据长度可能有8位或者9位，当采用9位长度时，需要额外的程序处理它

(5) 停止周期（通常是指停止位）的长度，可以是1、1.5或者2个标准位周期。尽管在低速数据传输中，每次传输过程都能容忍收发双方配置的数据位数不同，但是在高传输速率下这种不匹配就会导致数据传输错误

图4-6 简单的UART数据传输

由于采用校验增加了系统的额外软件开销，因此通常仅在接收端进行校验。在高噪声环境中，由于数据很可能被污染，因此校验显得特别有用。

位于数据字节末尾的停止位也很重要，这是因为它们能够保证传输线回到空闲态并保持足够的时间，以确保接收器同步硬件复位。通常情况下，为了提高数据链路的吞吐量，就希望停止位尽可能少，但是在噪声环境中或者有些节点需要更多的时间来处理所接收到的数据的情况下，就要增加停止位个数。

尽管UART主要用于两个PCB间的通信，但是在比这更大的传输距离下它也能可靠工作。对于单端RS-232链路来说，其典型的最大传输距离约为15m，而采用差分传输的RS-422和

RS-485链路则能达到4000m以上。UART接口的传输速率通常为300bit/s～115Kbit/s。

基本UART接口框架

有关UART接口的代码参见文件CommIf.c，其对应的函数声明和变量定义参见文件CommIFDef.h。该接口采用了基于缓存区的、中断驱动的框架，并具有环形接收区和发送缓冲区。通过利用缓存接口，该框架能使应用程序处理比UART内部4字节接收和发送缓存器更大的数据，这是时间临界型系统所必需的。此外，该接口还允许用户使用像dsPIC系列DSC这样的具有2个UART端口的芯片中的UART1或者UART2。 110

为了使用该接口，应用程序必须首先调用CommInit()函数以初始化希望使用的通信端口。该函数中的参数将指定初始化端口并配置相关运行参数，如通信速度、校验类型、停止位长度等。由于该函数负责初始化用于控制接口的全局状态变量，并配置UART硬件，因此在调用该初始化函数之前调用其他的UART 接口函数都会导致错误。CommInit()函数的代码参见代码实例4-1。 111

代码实例4-1 CommInit()函数

```
/***********************************************************************
*   FUNCTION:        CommInit(Uint8 ui8Port, Uint16 ui16BaudRate,          *
*                             Uint16 ui16Parity, Uint16 ui16StopBits)       *
*                                                                           *
*   DESCRIPTION:     This function configures the system's communication  *
*                    channel(s). It must be customized for the specific    *
*                    communication modules and channel parameters used     *
*                    for the particular application.                        *
*                                                                           *
*                    The function uses the global system communication     *
*                    configuration parameters; if any of these parameters  *
*                    are invalid, it uses the default channel parameters   *
*                    to ensure that the communication channel is at least  *
*                    operational.                                           *
*                                                                           *
*                    NOTE:   Although the dsPIC hardware allows 9-bit      *
*                            data, this routine only supports 8-bit data    *
*                            since that is more commonly used and is        *
*                            easier to manipulate.                          *
*                                                                           *
*   PARAMETERS:      ui8Port          - index of UART port to configure    *
*                                       (MUST be either UART_1 or UART_2)  *
*                    ui16BaudRate     - communication baud rate in bits/sec *
*                    ui16Parity       - type of parity to use (MUST be one *
*                                       of PARITY_*)                        *
*                    ui16StopBits     - number of stop bits (MUST be either *
*                                       STOP_BITS_1 or STOP_BITS_2)        *
*                                                                           *
*   RETURNS:         The function returns one of the following status      *
*                    code values:                                           *
*                    ST_OK            - operation successful                *
```

```
*                   ST_INV_PARM    - invalid communication parameter    *
*                                    detected, default channel          *
*                                    parameters used                    *
*                   ST_COMM_INIT   - failed to initialize the requested *
*                                    communication channel(s)           *
*                                                                       *
*   REVISION: 0    v1.0              DATE: 18 May 2006                  *
*       Original release.                                               *
***********************************************************************/

Uint16
    CommInit(Uint8 ui8Port, Uint16 ui16BaudRate, Uint16 ui16Parity,
             Uint16 ui16StopBits)
    {
    //  Local Variables

    Uint16
        ui16BRGValue,         // Baud Rate Generator value
        ui16Mode,             // Configuration data for UxMODE register
        ui16Status;           // Configuration data for UxSTA register

    //  Log the UART that we're using
    //  for the communication port

    g_ui8CommPort = ui8Port;

    //  Turn off the specified UART module and
    //  disable the associated interrupt so we
    //  can complete the initialization without
    //  interruption

    if (ui8Port == UART_2)
        CloseUART2();                         // Close UART 2

    else
        CloseUART1();                         // Close UART 1

    //  Compute the Baud Rate Generator value based
    //  on the system instruction cycle frequency
    //  and the desired baud rate. From the 30F6014A
    //  data sheet, the BRG value is computed as
    //    BRG = ((FCY / baud rate) / 16) - 1
    //  where
    //    FCY = instruction cycle frequency in Hz
    //    baud rate = desired baud rate in bits/sec
```

```
ui16BRGValue = ((FCY / ui16BaudRate) / 16) - 1;

//  Setup the UxMODE register value to specify
//  whether the UART is enabled, how it will handle
//  the IDLE state, and other configuration data

ui16Mode = UART_EN              & // Enable the UART
           UART_IDLE_CON        & // Stop the UART when in the IDLE state
           UART_DIS_WAKE        & // Don't enable UART Wake on Start
           UART_DIS_LOOPBACK &  // Disable loopback mode
           UART_DIS_ABAUD;        // Don't use Auto-baud mode

//  Add the appropriate configuration information
//  for the number of data bits and parity to employ

switch (ui16Parity)
    {
    case PARITY_EVEN:
        ui16Mode &= UART_EVEN_PAR_8BIT; // Even parity, 8 data bits
        break;

    case PARITY_ODD:
        ui16Mode &= UART_ODD_PAR_8BIT;  // Odd parity, 8 data bits
        break;

    default:
        ui16Mode &= UART_NO_PAR_8BIT;   // No parity, 8 data bits
        break;
    }

//  Add the appropriate configuration information
//  for the number of stop bits to use

if (ui16StopBits == STOP_BITS_2)
    ui16Mode &= UART_2STOPBITS;      // Use 2 stop bits

else
    ui16Mode &= UART_1STOPBIT;       // Use 1 stop bit
//  Setup the UxSTA register value to specify
//  how the transmit and receive interrupts
//  will be generated and other processing
//  control configuration

ui16Status = UART_INT_TX           &   // Interrupt on Tx register
             UART_TX_PIN_NORMAL    &   // Use normal Tx pin state (not
                                       //   transmitting a break)
```

```
                    UART_TX_DISABLE      &    // Disable the transmitter for now
                    UART_INT_RX_CHAR     &    // Interrupt on character reception
                    UART_ADR_DETECT_DIS  &    // Don't use Address Detect mode
                    UART_RX_OVERRUN_CLEAR;    // Clear the UART Rx Overrun flag

    //  Initialize the global state variables
    //  associated with the communication channel

    g_ui16CommRxAppIndex = 0;          // Initialize the app-side Rx buffer index
    g_ui16CommRxISRIndex = 0;          // Initialize the ISR-side Rx buffer index
    memset(g_ui8CommRxData, 0, COMM_RX_BUFF_SZ);  // Initialize Rx buffer data

    g_ui16CommTxAppIndex = 0;          // Initialize the app-side Tx buffer index
    g_ui16CommTxISRIndex = 0;          // Initialize the ISR-side Tx buffer index
    memset(g_ui8CommTxData, 0, COMM_TX_BUFF_SZ);  // Initialize Tx buffer data

    //  Actually initialize the appropriate UART

    if (ui8Port == UART_2)
        {
        ConfigIntUART2(UART_RX_INT_EN  & UART_RX_INT_PR6 &   // Enable Rx interrupt
                       UART_TX_INT_DIS & UART_TX_INT_PR3);   // and disable Tx int

        OpenUART2(ui16Mode, ui16Status, ui16BRGValue);  // Open UART 2
        }

    else
        {
        ConfigIntUART1(UART_RX_INT_EN  & UART_RX_INT_PR6 &   // Enable Rx interrupt
                       UART_TX_INT_DIS & UART_TX_INT_PR3);   // and disable Tx int

        OpenUART1(ui16Mode, ui16Status, ui16BRGValue);  // Open UART 1
        }

    return ui16Status;
    }
```

112~114

一旦利用CommInit()完成通信接口初始化，并且dsPIC系列DSC的中断优先级被设置得足够低以便能够响应串行端口中断（中断优先级6），那么应用程序就能从所选的UART中读写数据。为了从已初始化的串行端口中读取数据，应用程序首先应当通过调用CommGetRxPendingCount()函数检查全局接收数据队列中是否有待处理的数据，该函数的返回值代表了接收数据队列中待处理的数据字节数。如果返回值不为零，表示可以读取数据，于是应用程序就通过调用函数CommGetRxChar()从队列中读取下一个待处理数据。图4-7是整个处理过程的流程图。

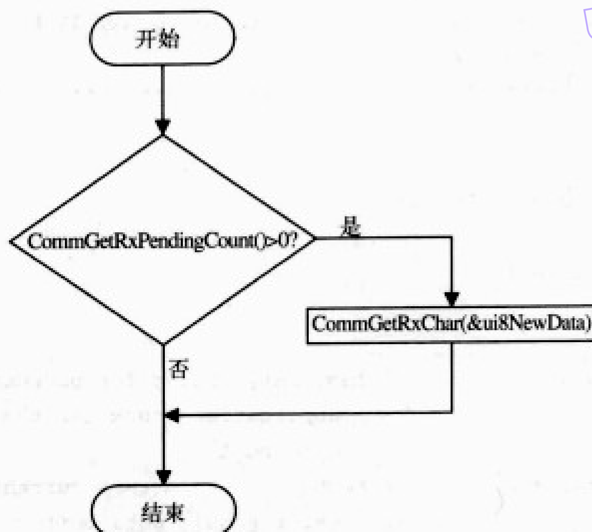


图4-7 应用程序读取UART数据的流程图

假设CommGetRxChar()函数返回的状态码为ST_OK，那么应用程序就可以放心地使用读自UART的数据。但是，如果状态码不是ST_OK，那么应用程序就不能使用ui8NewData中的数据，因为它是无效的。

传输数据的过程也很简单。假设已经调用CommInit()初始化了通信接口，那么应用程序就可以分别利用CommPutChar()和CommPutBuff()函数发送一个字节或者多个字节的数据。快速浏览一下CommPutBuff()的代码就会发现，其实它只是反复调用CommPut-Char()函数，从而将所有源数据放入环形发送缓存区。但是，研究CommPutChar()函数的代码（参见代码实例4-2）也是很有意义的。

115

代码实例4-2 CommPutChar()函数

```
/*************************************************************************
*   FUNCTION:       CommPutChar(Uint8 ui8Data)                           *
*                                                                        *
*   DESCRIPTION:    This function adds a character to the global        *
*                   Transmit Data buffer for subsequent transmission     *
*                   by the UART's Transmit Data interrupt handler.       *
*                   If the buffer is full, the character is not added    *
*                   to the buffer, and the function returns an error     *
*                   status.                                              *
*                                                                        *
*   PARAMETERS:     ui8Data - data to be added to the transmit data      *
*                             buffer for transmission by the USART       *
*                                                                        *
*   RETURNS:        The function returns one of the following status     *
*                   code values:                                         *
*                   ST_OK          - operation successful                *
*                   ST_BUFFER_FULL - transmit buffer full, data not      *
*                                    added for transmission              *
*                                                                        *
```

```
 *   REVISION: 0    v1.0                DATE: 18 May 2006                  *
 *       Original release.                                                 *
 ***************************************************************************/

Uint16
    CommPutChar(Uint8 ui8Data)
    {
    //   Local Variables

    Uint16
        ui16AppIndex,          // Temporary buffer for processing the
                               //   application index for the transmit
                               //   data buffer
        ui16DataCount,         // Number of characters currently in
                               //   the transmit data buffer
        ui16Status;            // Function execution status

    //   Check whether there is room in the
    //   transmit data buffer for the new data

    ui16Status     = ST_BUFFER_FULL; // Assume the transmit buffer is full
    ui16DataCount = CommGetTxBuffCount();   // Get the number of characters already
                                    //   in the transmit data buffer
if (ui16DataCount < (COMM_TX_BUFF_SZ - 1))
    {
    //   We do have room in the buffer,
    //   so add the new character to it

    g_ui8CommTxData[g_ui16CommTxAppIndex] = ui8Data;   // Add the new
                                                          character

    //   Modify the application-side index to point
    //   to the next character in the transmit buffer.
    //   Use a local variable to do the manipulations
    //   and then transfer value to the global state
    //   variable in one operation to prevent coherency
    //   problems with the ISR.

    ui16AppIndex = g_ui16CommTxAppIndex;      // Get the current
                                                application index
    ui16AppIndex++;                           // Move to the next slot
    ui16AppIndex &= (COMM_TX_BUFF_SZ - 1);   // Perform a rapid modulo-
                                             //     COMM_TX_BUFF_SZ
                                                    calculation
                                             //     (assumes that COMM_TX)
                                                    BUFF_SZ
```

```
                                    //   is a power of 2)

    //  Update the state variable that controls
    //  access to the global Transmit Data queue

    g_ui16CommTxAppIndex = ui16AppIndex;

    //  Make sure that the hardware can and will send
    //  the data. Since the data is actually loaded
    //  into the UART's Transmit Buffer in the ISR,
    //  we need to make sure that the ISR is called
    //  if the Transmit Buffer has room. If there is
    //  no room in the UART's Transmit Buffer, we don't
    //  need to generate an interrupt at this time (the
    //  new data has already been added to the global
    //  Transmit Data queue and will be loaded into the
    //  UART's Transmit Buffer at a later interrupt.)

    CommEnableTransmitter();    // Make sure the transmitter is enabled -
                                //   must occur FIRST
    if (CommTxBuffAvail())
        CommForceTxInt();       // UART's Transmit Buffer has room so
                                //   force a Transmit interrupt. This
                                //   must be performed BEFORE making sure
                                //   that the UART's Transmit interrupt
                                //   is enabled to avoid a race condition
                                //   with the Transmit Data ISR.

    CommEnableTxInt();          // Finally, ensure that the UART's
                                   Transmit
                                //   interrupt is enabled

    //  Flag that the operation was successful

    ui16Status = ST_OK;
    }

return ui16Status;
}
```

4.2.3 控制器局域网络

控制器局域网（CAN）通信接口可能是dsPIC系列DSC芯片上最为复杂的串行接口。它包含一个极为先进的内部硬件控制器，支持中速率（可达1Mbit/s）数据传输。它还集成了硬件错误检测功能和复杂的消息优先策略，并能设置滤波器只接收感兴趣的消息，所有这些功能都只需很小的处理器开销。CAN接口实现了在单个网络中链接多节点，已被广泛应用

于汽车工业和工业过程控制领域。

既然有这么多优点，那么为什么并不是所有系统都采用CAN接口呢？这主要有两个原因：复杂度和成本。CAN的最大优点是具有很强的配置能力，但这也正是其最大的缺点。因为它太灵活，CAN拓扑可以以相同的硬件用于各种应用。但这种灵活性必须通过精确的配置来实现，否则就会导致不可靠或者完全失效，并且排除故障时费时费力。

116~118

1. 基本的CAN架构

20世纪80年代早期博世公司开发的CAN架构极为简单。尽管CAN协议本身被有意设计成媒介中立的[①]，但是最常见的实现方式是采用一个差分串行总线连接两个或者多个节点，最高传输速率为1Mbit/s。算上对应的地信号，这种可靠的接口仅使用三根导线。

CAN通信协议属于CSMA/CD系列协议，该缩写表示载波监听多路访问/冲突检测（CSMA/CD）。尽管该系列协议的名称很长，但是其背后的概念却很简单。在一个*载波监听*系统中，所有的节点都会在准备发送信息前，监视网络是否能保持一段空闲期。一旦过了这段空闲期，网络中的任何节点都可以发送数据，这就是*多路访问*的含义。正如我们想到的，有时可能有两个或者多个节点同时要求发送数据（这种情况被成为*冲突*），因此网络必须具备*冲突检测*功能。CSMA/CD系列协议中各个成员都有不同的处理方式，但是相同类型（比如CAN总线）的所有成员处理方式都相同。

在这些任务（载波监听和冲突检测）中，冲突检测更困难一些。CAN的设计者提出了一种巧妙的解决方法，它允许系统设计师设定消息的优先级，以便较重要的消息比不太重要的消息能够更早地获得总线访问权（这与dsPIC系列DSC的中断优先级类似）。不仅CAN支持消息优先级，而且其网络*仲裁*[②]策略对更高优先级的消息也是*非破坏性的*[③]，并能保证高优先级的消息没有传输时延。消息仲裁非常重要，我们后面还会详细研究它。但是首先还需要了解一些背景知识。

CAN的另一个重要特性是它具有自建的错误检测电路，这种电路能标记出总线故障并且逐个去除网络中产生大量错误的节点。尽管该协议不支持错误检测，但是它的错误检测功能有助于避免某个产生错误的节点对整个网络带来的严重故障。但是，由于发生问题后会迅速积累错误，一旦节点停止发送，故障又会消失，因此很难定位故障源。

119

如果这些功能听起来会增加处理器的负担，那么请你放心。由于这些功能很复杂，因此大多数CAN接口都被集成在两个硬件元件中：一个是CAN总线控制器状态机，它负责处理所有的仲裁以及错误检测；另一个是CAN总线驱动器，它负责驱动和监控CAN总线的物理层。在大多数系统中，这两个元件被封装在独立的集成电路（IC）[④]中。dsPIC系列DSC也一样，其片上包含一个或者两个CAN总线控制器模块（与dsPIC器件的型号有关），但是还需要一个外置的CAN总线驱动器芯片以便连接到总线上。一旦CAN接口电路配置完毕，就能得到接收器里所有格式的数据消息和状态位，并能向其他节点发送完整的数据消息。由于所

---

① *媒介中立*是指协议对实现该协议所用的媒介不做特别要求。这是故意不指定的，以便协议能使用于各种物理传输媒介（只要传输媒介能够支持显性位和隐性位即可）。

② 这里的仲裁是指从两个或者更多个希望访问总线的节点中选出一个，只允许它发送数据。而中断仲裁则是指dsPIC系列DSC的中断控制器确定响应哪个中断的过程。

③ *非破坏性*仲裁是指最终送上总线的数据是完好无损的。而破坏性仲裁虽然也能确定出允许那个消息上总线，但是它会破坏消息，因此发送节点必须重新发送，这就增加了总的传输时间，并降低了可用的总线带宽。

④ 集成电路（IC）是一种包含系统大量电路的硅芯片。

有的错误检测与处理工作都是由硬件完成的，因此CAN接口的处理器工作量可以降到最低。

还有一个重要问题是CAN总线可以传输多远，答案是CAN总线的最大传输距离与总线支持的数据传输率有关。表4-1[①]显示了推荐的在不同位速率下的最大总线长度。

表4-1 推荐的CAN总线最大长度

位速率（Kbit/s）	总线长度（m）
1000	30
500	100
250	250
125	500
62.5	1000

从表4-1中可以清晰地看出，总线最大长度随着数据传输速率的增大而迅速减小，但是，即使在1Mbit/s的速率下，最大总线长度也是留有裕量的（相当稳健的）。

2. CAN数据格式

根据CAN2.0协议[②]，CAN总线上发送的数据共有如下4种形式（称为帧）。

120

（1）数据帧，从一个节点向总线上的其他所有节点发送数据。

（2）远程传输帧，向总线上的另一个节点请求数据。

（3）错误帧，报告检测到的通信错误。

（4）过载帧，报告传输节点正在忙于处理之前的消息，并且此时无法接收更多的数据。

在我们的例子中，主要研究最常用的帧格式——数据帧，它又可分为两种：标准帧和扩展帧。这两种数据帧格式参见图4-8a和图4-8b，它们本质上是相同的，唯一的实际差别是标准帧的仲裁ID更短。所有的数据帧格式具有如下的基本部分。

（1）仲裁ID域，其大小与帧类型有关。

（2）控制域，长度为6bit。

（3）数据域，长度为0～8B。

（4）CRC域，长度为2B。

（5）应答域，长度为2bit。

（6）帧结束标志，长度为1bit。

在这些域中，用户可以配置仲裁ID域、控制域以及数据域，而CAN控制器的硬件会自动产生并确认CRC域、应答域以及帧结束标志。在研究CAN的仲裁技术前，还是先深入了解一下这些域。

在标准数据帧中，仲裁ID域包括11位标识符以及1位远程传输请求（RTR）标志位。扩展数据帧与此有所不同，但是如果标准帧与扩展帧发生冲突，那么标准帧的优先级更高。对于一个扩展帧来说，标识符共有29位，其中最高的11位在帧起始位之后立即发送，这之后是1位替换远程请求标志位（SRR）、1位表示扩展标志位（IDE）、剩余的18位标识符和1位RTR标志位。

① 该表引自Microchip Application Note 713 – Controller Area Network（CAN） Basis，读者可以从Microchip公司的网站上获得（文档编号DS00713A）。

② CAN Specification 2.0，Robert Bosch GmbH，1991。

CAN总线的标准（11位ID）数据帧格式



要点

（1）每个消息包括4个主域和一些域标志位

(a) 12位仲裁ID域

(b) 6位控制域

(c) 8N位数据域，N为字节数

(d) 16位CRC域

用户只能配置前3个域，CAN总线控制器的硬件会自动配置CRC域和帧标志位

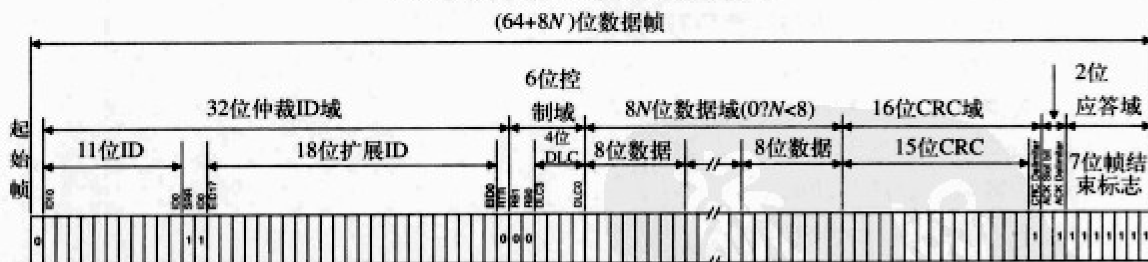
（2）值为1的位表示隐性态，值为0的位表示显性态；

（3）总线仲裁将根据仲裁ID域的值决定哪个节点可以发送数据，显性态数据排第一的消息掌握优先权，并可以被传上总线。实际中，这意味着域值越低，信息的优先级越高

（4）标准数据帧比扩展数据帧的优先级高

(a) CAN总线的标准数据帧格式

CAN总线的扩展（29位ID）数据帧格式



要点

（1）每个消息包括4个主域和一些域标志位

(a) 32位仲裁ID域

(b) 6位控制域

(c) 8N位数据域，N为字节数

(d) 16位CRC域

用户只能配置前3个域，CAN总线控制器的硬件会自动配置CRC域和帧标志位

（2）值为1的位表示隐性态，值为0的位表示显性态

（3）总线仲裁将根据仲裁ID域的值决定哪个节点可以发送数据，显性态数据排第一的消息掌握优先权，并可以被传上总线。实际中，这意味着域值越低，信息的优先级越高

（4）根据总线仲裁策略，由于标准数据帧的0值RTR位（显性态）与扩展数据帧的1值SRR位（隐形态）对齐，因此标准数据帧比扩展数据帧的优先级高

(b) CAN总线的扩展数据帧格式

图　4-8

尽管这两种数据格式的控制域长度都是6位，但是它们还略有差别。在标准数据帧中，控制域的第一位是IDE标志，紧随其后的是一个保留位（在CAN规范中被记作“r0”）。控制域的其他4位是数据长度代码（DLC），它表示本消息中包含的数据字节数。尽管DLC有4位，但是由于协议支持的每个消息能够传输的最大数据长度为8字节，因此它的取值范围只能是0～8。由于扩展帧的仲裁ID域中包括IDE标志，因此控制域中有两个隐性位：“r1”和“r0”。标准数据帧中的DLC与此类似，并且受到相同的约束。

121～122

循环冗余码（CRC）域是由硬件自动配置的，并且对程序员来说是透明的，设计师不必太关心它。为了叙述完整起见，请注意CRC值本身只有15位，而CRC域则由CRC值和1位CRC定界位组成。

CAN消息中的最后一个域是应答（ACK）域，其长度为2bit。其中包括一个前导的ACK间隙位，发送节点利用该位来设定隐性态（将在下一段给出其定义），而所有成功接收该信息的节点又可以利用该位来设定显性态。ACK域的最后一位（也是整个信息的最后一位）是ACK定界符号位，它只是使总线返回隐性态，表明传输完毕。

尽管在我们的例子中不使用远程帧、错误帧以及过载帧，但是dsPIC系列DSC的CAN总线接口完全能够处理它们。远程帧用于请求从某节点自动传输数据（在发出请求前数据已经载入CAN模块），而错误帧则由检测到总线错误的节点发出。由于错误帧会有意扰乱CAN总线的时序，因此它们会导致所有正在发送数据的节点都停止发送，待复位后再重新发送。

3. 总线仲裁

前面已经提到，当采用异步方式传输数据时，假如有两个节点同时希望发送数据，就需要采用一些访问仲裁策略来决定允许哪个节点发送。CAN总线的设计师采用了一种巧妙的解决方法，它采用了一种非破坏性的仲裁策略，利用冲突信息的仲裁ID值来决定哪个节点拥有优先权。为便于理解该策略是如何工作的，我们首先需要学习有关CAN总线的两个名词。CAN总线上的数据可分为显性态（逻辑0）或隐性态（逻辑1）。当同时传输不同状态的两位时，显性态“获胜”，即它是总线上的实际状态。

CAN总线就是利用这个原理实现传输范围的仲裁的。无论是两个或者多个节点同时打算传输信息，总线上出现的总是显性态。由于每个节点每次只向总线发送1bit数据，因此可以检查总线上的数据与最近发送的状态位是否相同。如果发送节点发送了一个隐性态，但是检测到总线上是显性态，那么该节点就知道还有另一个节点在发送数据，而发送隐性态数据的节点就将离开总线。隐性节点会立即停止发送并且等到当前的发送过程结束后再发送自己的数据。

123

通过以这种方式处理仲裁，CAN总线就能以一种结构性的方法来发送数据，即使发生冲突也不会导致数据丢失，并且所有节点都会重复发送各自的信息。由于显性态为0，而基于CAN总线的系统的设计师可以选择仲裁ID使最重要的信息具有较低的ID值，从而具有最高的优先级。例如，可以选择仲裁ID值000H～01FH作为报警条件，而ID值020H～7FFH作为普通工作信息，设计师要保证报警信息比普通工作信息的优先级更高。在图4-9所示的例子中，同时发送了仲裁ID值为010H报警信息和仲裁ID值为040H的普通工作信息。此外，该策略还允许总线上同时出现标准和扩展数据帧，其中标准帧的优先级比扩展帧高。

4. 接受滤波器

CAN协议的一个可选功能是所有的CAN控制器都具有消息滤波功能，这可以使控制器

只接受那些仲裁ID域与可编程位映射滤波器匹配的消息。这里，当我们提到滤波器时，并不是指处理数字信号的数字滤波器，而是指一个过程，在该过程中CAN控制器只选择满足一定条件的消息来处理。请注意，即使控制器选择屏蔽某消息，但是控制器仍会准确地响应应答间隙位（ACK Slot bit）。滤波是一种能够降低处理器开销的中级处理技术，是通过限制希望处理的消息类型实现的，而应答是底层处理要求的，它能保证所有节点间都能精确传输。

124

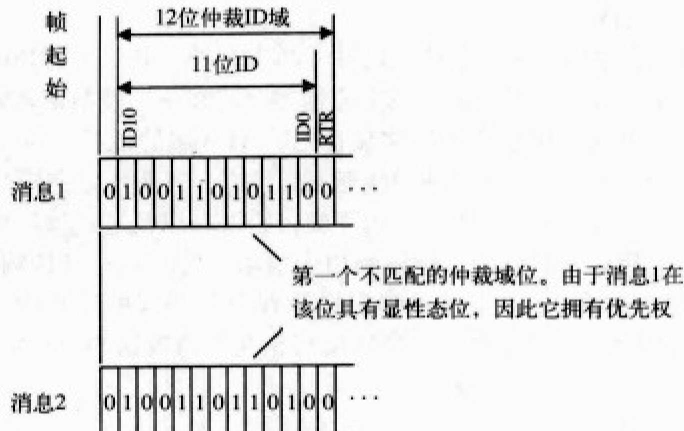


图4-9　两个同时发送的消息的仲裁例子

滤波CAN的消息分为两步，它们都由设计师配置，并由CAN控制器的硬件执行。首先，我们需要配置接受滤波器的值（dsPIC系列DSC支持6个不同的滤波器），它将和接收到的消息的仲裁ID位进行逻辑位与操作。计算结果再与接受屏蔽字比较，如果该结果与接受屏蔽字匹配，那么接收到的信息将被添至CAN总线接收缓冲区（假设缓冲区中还有空间）。

这一点对于CAN应用系统的设计新手来说很容易混淆（有时有经验的设计师也会出错），因此下面将举例说明。假设我们希望接受仲裁ID域值范围是300H～3FFH的所有标准CAN数据帧。那么，接受滤波器就应配置为F00H，而接受屏蔽字应为300H，由于接受滤波器和接收到的消息的12位仲裁ID域的逻辑与运算，这会使仲裁ID域的低字节不在乎任何条件（由于整个低字节都被“与”为0了），因此滤波器会允许上半字节通过。只有上半字节的仲裁ID域值等于3，才能和接受屏蔽字匹配，对应的消息才会被接受。

5. 基本的CAN接口框架

Microchip公司的16-bit Peripheral Library提供了程序员需要的与dsPIC系列DSC的CAN控制器接口的极好工具，但是和UART接口一样，它们并不会直接实现中断驱动的缓存消息I/O（由于CAN总线是基于消息的协议，因此这不是基本特征）。幸运的是，我们只需简单地修改之前为UART设计的代码，就能得到适用于CAN总线的类似框架。这是一个很好的例子，

125

它体现出编写代码的通用性的价值，由于UART和CAN总线所采用的基本原理相同，因此我们可以在CAN总线设计中重用为UART设计的代码（当然，还是要针对两个模块的区别进行必要的修改）。

4.3 高级协议

到目前为止，我们将基本的通信信道都视为I/O，利用它们在dsPIC系列DSC和其他器件间传输数据，并且我们的焦点还集中在位级和字节级。即使在探讨CAN总线协议时，我们真正研究的还是底层的硬件接口以及适当配置它的初始化信息。这些都是必需的，但是还不够，我们还需要定义所传输的信息代表的真实含义，因此我们从现在起要转向下一个话题——高级协议，或者记为HLP（high-level protocol）。

前面几章已经讨论了将数据转换为信息的必要性，并且还能够利用基于高级协议搭建的结构精确地处理它们。这种情况与一个人向另一个人发信类似。为了使信件到达目的地，我们已经指定了一些信息（收件人地址）并且我们还可以根据情况增加可选的信息（比如收件人的姓名）。由于传输过程可能出现问题，最好还有用于报告故障和故障恢复的信息（比如，发信人的姓名和地址）。这些都是我们传输数据时必须处理的内容，无论是在系统中的两个节点间传输还是在不同设备（比如dsPIC系列DSC与外部DAC或dsPIC系列DSC与远程系统）间传输，都得考虑。通过提出一种合适的数据传输架构，可以更加清晰地理解数据，并能更加有效地处理它们。

很多流行的标准协议都适用于串行数据，根据传感器必须与之通信的设备类型，设计师可能在某个应用中别无选择地采用某个协议。如果某个协议应用得很广泛，那么就可能（或者至少有一定的概率）有第三方的软件实现，我们可以直接将其用于自己的设计，并且用它们来测试结果代码。后一种情况可能出现在某个应用只需要第三方软件的部分功能来完成其任务时，因为实现标准协议的全部功能会浪费大量的处理器资源。然而有时，这却是唯一的选择——比如利用SPI接口与另一个芯片通信。通常，每个设备都必须采用特殊的协议才能与之通信，这会使设计师只有少量的设计灵活性或者根本没有。

126

有一种替代的方法是为某个应用设计一套专用协议，该方法被用于主机系统和dsPIC系列DSC通信的实例中。通过适当的设计和合理的实现，该专用协议可以提供所需的全部功能，并且不会因为那些系统从不会用到的功能而浪费资源。但是，设计师最好明白，对于应用设计而言，真正需要考虑的是（也是开发小组能够承担的）开发时间以及开发专用解决方案所需的资源[①]。

如果在客观地研究了各种情况后，设计师认为必须采用专用协议，那么就要使其尽可能地轻便、可靠和可扩展。轻便是指在完成任务时要尽可能少地增加处理器的开销。尽管可靠性的概念一目了然，但是我们还应考虑所需的错误检测功能。协议的可扩展性是指，它支持方便地增加新的消息类型并且对已有代码的影响非常小。

下面例子中所采用的协议就展示一个这种实现（当然还有其他的），它能兼顾处理开销小、错误检测功能以及可扩展性等要求。读者应当认识到，对于某个应用来说，本例并不是唯一的方法，甚至可能还不是最好的方法。但是，它的确能在各种环境下较好地工作。实际上，这些例子中有两个协议已经用于实际系统，一个是概要性的通用信息协议，另一个是特殊消息协议。下面，首先介绍通用消息协议。

① 很多工程师（以及公司）都患有“与我无关综合症”（Not Invented Here Syndrome，NIHS），这是指，只要不是工程师自己或者是他们公司设计的东西就一概不理。尽管通常是幽默地谈及该话题，但是其后果却一点儿也不好笑。它会导致开发周期严重增加、错失市场机遇以及得到劣质的产品。这会浪费公司大量的资金并损失利润，因此必须基于客观的事实来决定开发一个产品，而绝不能凭一时的冲动。

4.3.1 通用消息协议

通用消息协议是一种简单的命令/响应协议，它的主要目的只有一个：将命令消息传送至目标系统，并能处理返回的响应消息，其中，命令消息和响应消息的长度都是可变的。图4-10和图4-11所示的简单协议就能完美地处理上述任务。

起始标志 (1字节)	命令ID号 (1字节)	数据长度 (1字节)	数据 (N字节)	校验和 (1字节)
0x01(SOH)	—	N	—	计算

127

图4-10 通用消息协议的命令消息格式

对于一个命令来说，主要包括以下几个部分：

起始标志 命令的起始标志（起始头的ASCII码，0x01）

命令ID号 发送的命令ID号（最高有效位置位表示命令）

数据长度 命令对应的数据的长度（N字节，其中$0 \leqslant N \leqslant 255$）

数据 N字节命令数据（如果N==0不使用）

校验和 由命令ID号、数据长度以及所有数据计算得到的校验和

命令响应消息的格式参见图4-11。

起始标志 (1字节)	响应ID号 (1字节)	状态 (1字节)	数据长度 (1字节)	数据 (M字节)	校验和 (1字节)
0x01(SOH)	—	—	M	—	计算

图4-11 通用消息协议的响应消息格式

其中，

起始标志 命令的起始标志（起始头的ASCII码，0x01）

命令ID号 返回的响应ID号（等于对应的命令ID号，但是最高有效位被清零以表示响应）

数据长度 命令对应的数据的长度（M字节，其中$0 \leqslant M \leqslant 255$，$N$和$M$不一定相等）

数据 M字节响应数据（如果M等于0不使用）

校验和 由命令ID号、数据长度以及所有数据计算得到的校验和

由于可以假设通信时的错误出现在同一个点，因此该协议通过起始标志来表征新一轮的传输，并用校验和[①]来验证接收到的数据是否与发送的数据相同。这不仅提供了一定的错误检测功能，而且在出现错误后，接收器可以查找出起始标志，从而实现发送器和接收器的再同步。计算校验和的方法有很多种，本例是通过计算起始标志和校验和之间的所有数据的和，然后对结果做位取反操作得到。将和取反而不是用和本身的好处是：长串0数据的校验和是FFH而不是00H，从而使得校验和与数据本身有所区别。如果它们没有区别，那么由于数据和校验和的值都是0，因此停留在0的信号就无法被检测到。

128

为了进一步明确标记出命令消息，命令ID的最高有效位都被置1，而对应的响应ID与命

① 校验和是一种在数据接收端用于校验所接收数据是否正确的常用方法。之所以这么叫，是因为通常是计算传输的每个字节的数据之和作为校验和的，而计算结果又会与传输后的计算值做比较。

令ID一样，但其最高位则被清零。这样做是为了便于设计师进行通信调试，因为他们能在数据分析仪上很容易地区分出命令消息和响应消息。此外它还表明命令消息已被正确接收到。

正如我们前面讨论过的，当通过状态机实现协议时，要确保状态机决不会进入一个无法恢复的状态，这一点至关重要。这还能避免我们做出错误的假设，比如"消息总有特定的长度"。尽管我们总希望消息的长度是固定的，但是有时接收到的数据可能就不是预计的长度，而我们又不能因为它而停止通信，这样就会发生严重错误。图4-12显示了我们使用的协议分析状态机的流程图。

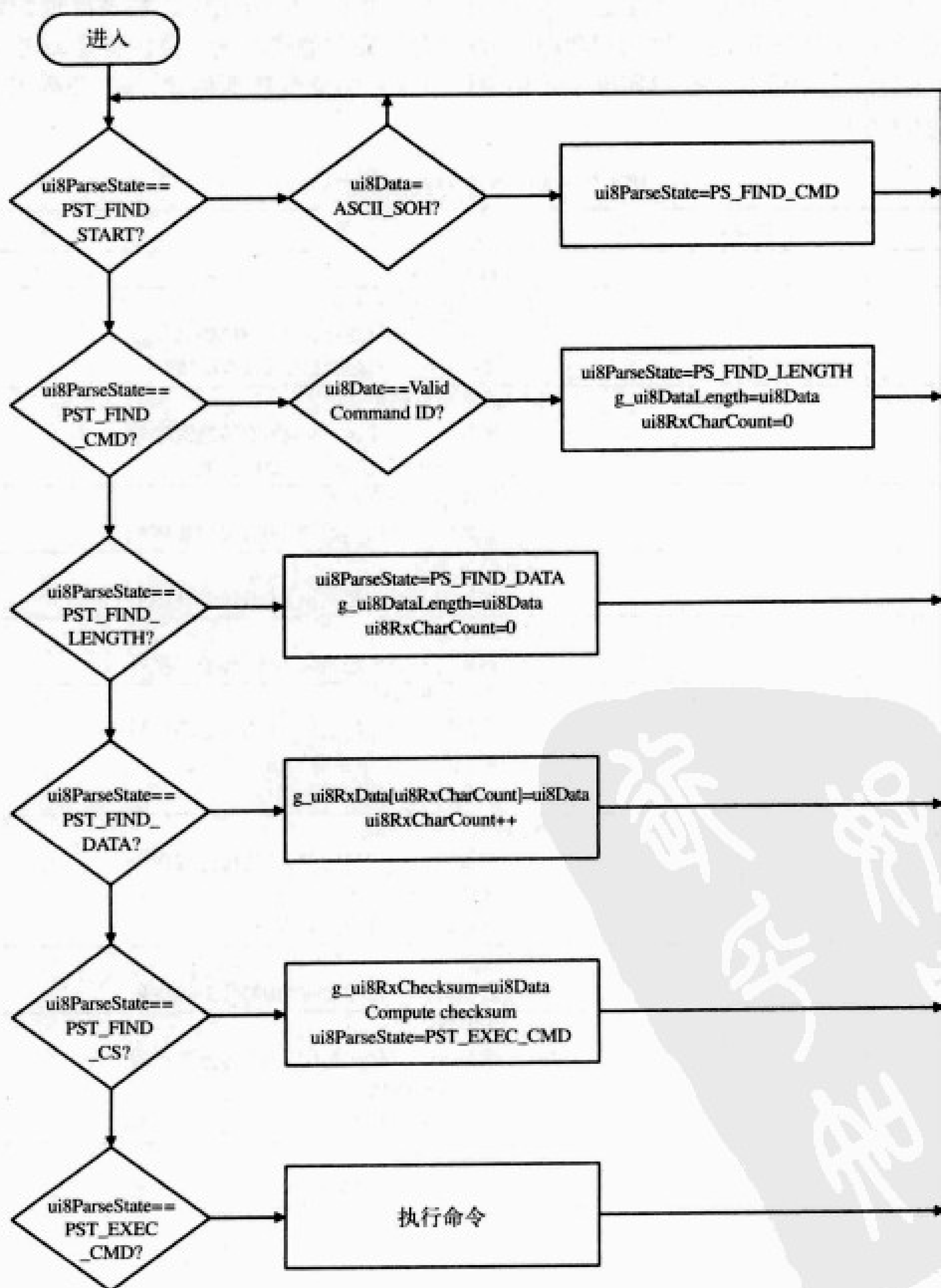


图4-12 处理协议的状态机

4.3.2 特殊命令协议

每个命令都有自己的一套数据需要发往接收器，而接收器在处理完命令消息后返回的响应消息也是与命令有关的。尽管特定的命令和响应有时是与应用有关的，但是这里给出对本书后面介绍的三种设计应用都适用的形式，以便给读者提供了一个实现的例子。请注意，特定命令协议也有错误检查功能。它利用通用消息系统来保证向接收器传送正确的数据。一旦数据到达，就认为是正确的。当然，用于分析数据的函数将检查这些数据是否适合特殊的任务。例如，如果主机发送了一个设定1500°F的温度的命令，但是允许的最高温度设定点为1000°F，那么系统能够正确接收到1500°F（即与发送器传输的值相同），但是却是无效的（这是一个无效的参数值）。表4-2列出了示例应用中常见的命令数据格式，表4-3列出了对应的响应消息的格式。

129~130

表4-2　命令消息的数据格式

命令ID	数据长度	参　数　值
0x80	0	报告版本信息
0x81	5	记录的低位校准测量值 参数0——8位起始位为0的传感器序号 参数1——32位定义的低位校准值
0x82	5	记录的高位校准测量值 参数0——8位起始位为0的传感器序号 参数1——32位定义的高位校准值
0x83	1	计算校准增益和偏置值 参数0——8位起始位为0的传感器序号
0x84	1	报告校正增益 参数0——8位起始位为0的传感器序号
0x85	1	报告校正偏置值 参数0——8位起始位为0的传感器序号
0x86	6	配置低位报警阈值 参数0——8位起始位为0的传感器序号 参数1——32位报警阈值 参数2——8位报警启用位
0x87	6	配置高位报警阈值 参数0——8位起始位为0的传感器序号 参数1——32位报警阈值 参数2——8位报警启用位
0x88	1	复位报警 参数0——8位起始位为0的传感器序号
0x89	1	报告报警状态 参数0——8位起始位为0的传感器序号
0x8A	1	报告低位报警阈值 参数0——8位起始位为0的传感器序号
0x8B	1	报告高位报警阈值 参数0——8位起始位为0的传感器序号
0x8C	1	报告传感器值 参数0——8位起始位为0的传感器序号
0x8D	1	设置数字电位计值 参数0——8位数字电位计值

表4-3 响应消息的数据格式

响应ID*	数据长度	参数值
0x00	6	报告版本信息 参数0——8位版本时期：月 参数1——8位版本时期：日 参数2——8位版本时期：年（高2位） 参数3——8位版本时期：年（低2位） 参数4——8位主版本ID号 参数5——8位副版本ID号
0x01	3	记录的低位校准测量值 参数0——8位起始位为0的传感器序号 参数1——校正点测量值的高2字节 参数2——校正点测量值的低2字节
0x02	3	记录的高位校准测量值 参数0——8位起始位为0的传感器序号 参数1——校正点测量值的高2字节 参数2——校正点测量值的低2字节
0x03	3	计算校准增益和偏置值 参数0——8位起始位为0的传感器序号
0x04	5	报告校正增益 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（增益值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（增益值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（增益值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（增益值）的低字的低字节
0x05	5	报告校正偏置值 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（偏置值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（偏置值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（偏置值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（偏置值）的低字的低字节
0x06	3	配置低位报警阈值 参数0——8位起始位为0的传感器序号 参数1——32位报警阈值 参数2——8位报警启用位
0x07	6	配置高位报警阈值 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（偏置值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（偏置值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（偏置值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（偏置值）的低字的低字节 参数5——8位报警启用位
0x08	1	复位报警 参数0——8位起始位为0的传感器序号

131
~
132

（续）

命令ID	数据长度	参数值
0x09	3	报告报警状态 参数0——8位起始位为0的传感器序号 参数1——2字节位映射报警标志的高字节 参数2——2字节位映射报警标志的低字节
0x0A	6	报告低位报警阈值 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（报警阈值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（报警阈值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（报警阈值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（报警阈值）的低字的低字节 参数5——8位报警启用位
0x0B	6	报告低位报警阈值 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（报警阈值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（报警阈值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（报警阈值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（报警阈值）的低字的低字节 参数5——8位起始位为0的报警启用位
0x0C	5	报告传感器值 参数0——8位起始位为0的传感器序号 参数1——4字节IEEE-754格式浮点数（传感器值）的高字的高字节 参数2——4字节IEEE-754格式浮点数（传感器值）的高字的低字节 参数3——4字节IEEE-754格式浮点数（传感器值）的低字的高字节 参数4——4字节IEEE-754格式浮点数（传感器值）的低字的低字节
0x0D	1	设置数字电位计的值 参数0——8位数字电位计的值

133

4.4 小结

本章内容涉及面较广。主要研究了dsPIC系列DSC与片外外围设备、传感器网络、控制器以及远程系统相连时常用的三种不同的通信接口。尽管读者可能不觉得，但是我们对CAN接口的介绍还很浅显（而本系列器件的参考手册花费了73页的篇幅来介绍这个模块），这里只是向设计者展示了可用的通信手段，并提供了丰富的适用于与传感器相连的标准或专用协议。掌握了这些工具，设计师就具备了扎实的技术基础，在此基础上就能建立并扩展基于dsPIC系统的通信能力。

134

第5章 dsPIC系列DSC的基本开发工具

通过前面两章的介绍，我们已经基本了解了dsPIC系列DSC的功能模块。本章将构建一个基本的软件模块开发包，利用它们就能建立灵活而通用的软硬件框架，并应用于智能传感器的实现。我们在应用开发中将使用很多Microchip公司免费提供（或者价格极低的）的软件工具和硬件开发平台。尽管还有很多第三方提供的好用的开发工具，但是Microchip公司提供的工具都通过了三种重要的测试——即时可用、价格便宜并且性能良好，因此是我们的首选。

5.1 应用测试平台

实际的产品开发中最令人头疼的情况是在新的未经测试的硬件上尝试新的未经测试的软件。于是，代码或者硬件中的任何错误都能导致系统突然崩溃，而且难以确定故障源。为了避免发生这种问题，我们将采用Microchip公司的dsPICDEM1.1通用开发板（GPDB，见图5-1）[①]来开发应用系统，在该开发板上可以方便地使用dsPIC30F6014A芯片。当然，在大多数情况下，考虑到具体应用的成本、尺寸、功耗以及功能要求等因素，最终产品会采用定制的硬件，但是先在标准的、已知完好的硬件平台上进行开发，可以在最终系统硬件准备就绪前就开始编程，并能在调试时避开一个主要的故障源（硬件）。

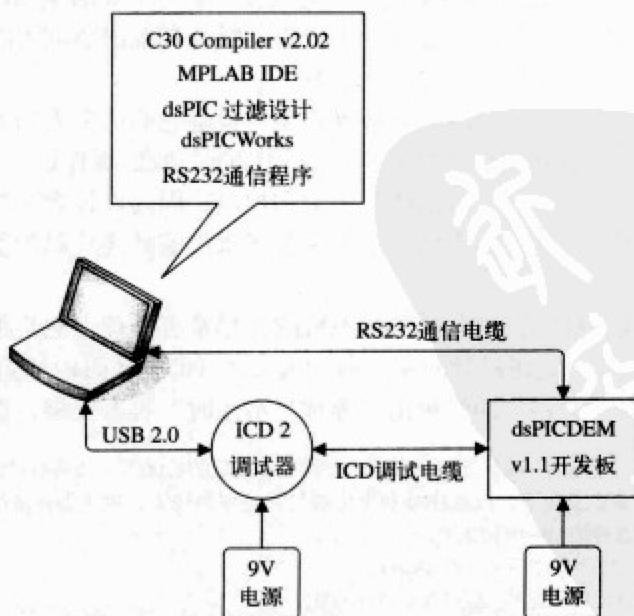


图5-1 dsPIC测试平台的框图

① Microchip公司的dsPICDEM 1.1通用开发板的产品编号是DM300014。

此外，我们还需要一些工具用于在通用开发板上测试代码，并且任何重要的软件开发都需要一些工具来检查应用程序是否工作正常。利用Microchip公司的ICD 2在线调试器和MPLAB IDE（集成开发环境），我们就能完成上述任务。MPLAB提供了方便的基于PC的软件开发环境，它包括编辑器、仿真器和编译器。尽管还有很多针对dsPIC系列DSC的很好的C编译器，但是这里所编写的代码都是面向Microchip C30 Compiler v2.02的，从Microchip公司的网站[①]上可以免费下载到它的学生版。C30编译器的优点之一是它可以直接集成到MAPLAB IDE中，当然，其他一些好的编译器也可以这样。ICD 2[②]必须购买，而MPLAB IDE[③]可以从Microchip公司的网站上免费下载。

5.2　固件框架概览

我们已经搭建了测试平台，下面就要建立固件框架，它是本书后续应用开发的基础。在开发代码时，我们将采用如下的设计理论以确保代码的稳健性，以便能预计出各种正常及异常的情况，并能应对它们。

（1）代码将采用软件状态机，它带有错误检测和相应的自恢复功能。这些软件状态机将尽可能独立而且简单，并且和应用系统中的其他软件之间有定义完整、功能正确的接口。

（2）为了尽可能扩展，固件采用Microchip公司提供的标准代码库，以节省应用代码的开发和调试时间。由于我们采用了已经测试过的代码[④]，因此还能简化维护工作。

（3）分层次地编写代码，特别是对于通信接口和其他访问底层硬件模块的代码。这有助于防止部分代码的变化影响其他、无关部分的代码，从而使设计者在开发代码模块时拥有最大的编程灵活性。

（4）所有代码都应有完整的内部文档，并有大量、有意义的注释和标准化的命名约定，以便于今后的开发人员维护系统。尽管本书中代码片断受篇幅限制都去除了很多注释，但是在本书附属资源中的实际代码都含有完整的注释。

这些设计理论中，第一条和最后一条最为重要。但是它们经常因为太耗费时间或者不是必需的而被忽视。在实现自纠错软件状态机处理过程时，我们要保证（至少要尽最大努力）代码决不会进入无法恢复的状态。这需要想象力和规范，因为设计者必须预计所有可能的异常条件并设计能够检测并且恢复的方法，但是它会严重影响最终代码的稳健性、灵活性以及可维护性。

设计高效的软件状态机的关键之一是状态机的文档是否合理，它将指引我们得出更为通用的代码文档。一些程序员觉得没时间写*内部代码文档*（即源代码内部固件的文档），经常不写或者一点也不重视这些文档，同时承诺“等项目结束时”再写文档，而这个承诺未必会履

---

① Microchip公司的网址是www.microchip.com。学生版在安装后的前60天支持所有功能，之后只支持最少的代码优化功能。在大多数情况下，以这样级别优化的代码长度会较大，但是编译器的功能不变。支持全部功能的编译器的产品编号为SW006012。

② Microchip公司ICD 2的产品编号为DV164005。

③ Microchip公司MPLAB IDE的产品编号为SW007002。

④ 和对待其他开发工具一样，明智的设计师会对“已测试的代码”略有怀疑。尽管很多程序员在遇到问题时很可能会立即认为“肯定是别人的代码有问题”，但是更好的方法是推测那些自己编写的未经测试的代码有问题。如果自己编写的代码经过彻底检查，并且发现没有错误，那么就应该检查第三方“已测试”的代码是否工作正常（要检测所有因为习以为常而忽略的步骤里，确保对写入测试代码的参数是正确的）。请记住，尽管这些软件“已测试”过，但是并不意味它们在出现问题的特殊条件下被测试过。

行。即使在完成固件设计后再添加注释，它们也缺乏在编写代码时添加的注释所具有的洞察力，因为那时候程序员清楚地明白为何要这样编写代码。事实上，在编写代码时不写文档就能充分体现该程序员极其不负责任，并且说明他要么太懒惰、要么太自私了（因为他认为没有注释别人就不知道他是如何做出来的），这样的人不适合从事产品开发。合理的代码文档可以减少将来的代码维护工作量，降低将来修改代码时忽略隐含要求（但在文档中未说明）的几率，从而数倍地节省投资。无论何时，如果有人对你说注释无关紧要，你就可以确信他确实还不懂得应该如何开发软件。程序员绝不能因为时间紧、任务重等压力就粗制滥造一些无注释的代码。毕竟，当（不是如果）这些缺少注释的代码出现问题时，他们就会名声扫地。

以这些设计理论作为基础，我们就将注意力转向应用程序的框架本身。为了便于理解，将从两方面解释框架：一个是贯穿于应用中的数据流，另一个是各种系统部件之间的时序关系。这样会使读者清晰地掌握是以何种方式将原始信号数据转换为有意义的信息，并且是如何一步一步地实现这种转换的。我们的分析将首先从研究应用中的数据流开始。

5.2.1 应用程序的数据流

数据流是以一种非常直接的方式贯穿于应用程序框架中的，具体参见图5-2。原始的模拟传感器信号首先通过传感器专用模拟信号调理电路，该调理电路能限制信号中的频率成分，使之满足奈奎斯特采样定理的要求，并将模拟电压的幅度调整到适当的范围。然后，ADC模块对调理后的信号进行数字化，并将数字化结果存储在一个与数据分析软件状态机共享的软件缓冲器中。待数据分析模块处理完数字化数据后，它会将处理结果传入另一个共享缓冲器，该缓冲器是控制过程状态机以及数据报告状态机的输入接口。控制过程模块会根据应用程序指定的算法更新控制输出量，而数据报告模块则负责将分析后的数据发送至外部的系统部件以便它们使用。 140

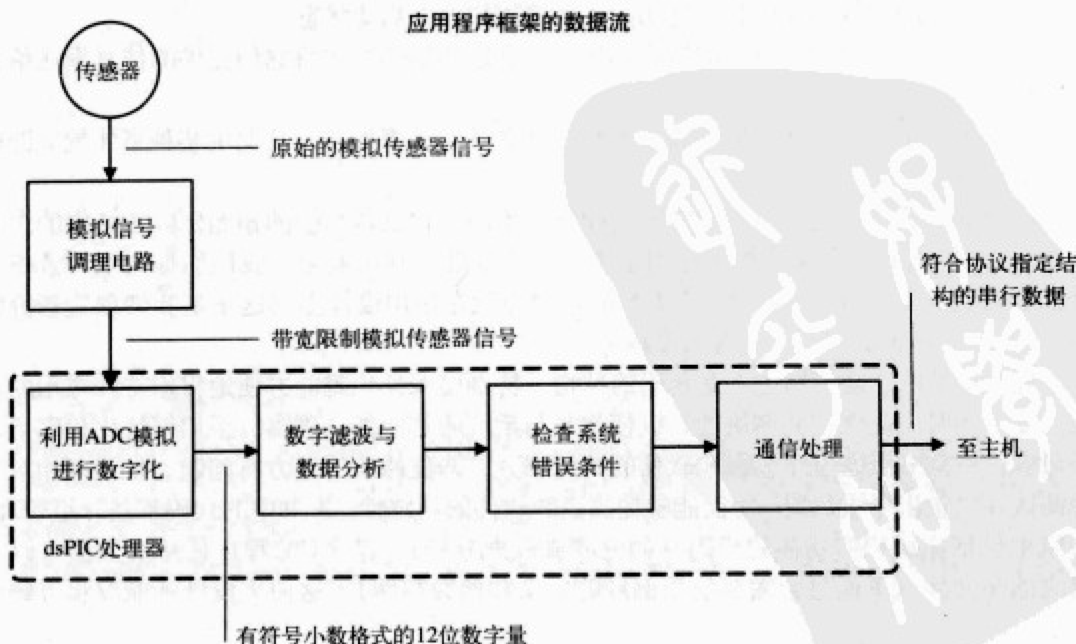


图5-2 通用的应用程序框架的数据流

所有这些都是按部就班地进行着，数据以期望的时间间隔依次进入系统，并按照要求被送往控制算法和通信接口，然后下一次采样结果又进入系统。这不是说控制算法和通信接口的工作频率必须和输入信号采样速率相等（通常它们都不是如此），但是输入和输出速率间的确存在一种直接的、通常还是常数的关系。例如，某个应用中对输入信号的采样速率可能是每秒1000次（1ksps①），而生成的分析数据的速率只有每秒250个，而控制算法和通信接口可能还工作在更低的速率上。关键问题是，每个处理级的输入量都是以前一级的输出速率被更新的，因此它们必须在新数据到来前就准备就绪。

有一种数据流是异步的，这就是错误报告。这些错误条件能反映出来自传感器的输入信号的故障（例如，信号值超出了预期的范围）、基本硬件的错误、算法错误或者只是固件检测到的有效但是不期望出现的条件（比如导线断开）。无论是何种原因，一旦发生异常的工作条件，程序框架都应报告，以便系统中其他依赖于传感器数据的部件可以采取适当的纠错措施。错误处理的异步性可以从每个主要处理模块以及报告模块最后发出的“错误处理”消息中看出。

141

请注意，报告模块和外部系统部件之间的通信链路可以是任何一种串行通信接口：比如UART（标准的RS-232或者RS-485链路）、CAN总线、SPI总线或者I²C总线（当距离很短时使用，如板到板）。尽管报告数据的实际结构取决于应用要求和通信信道的限制，但是报告系统的数据和异常条件这一高级概念适用于所有通信方式。

5.2.2 系统任务流

在分析了通过框架的数据流之后，下面将研究，为了合理完成任务而需要实现的数据流和时序要求的各种软件任务。实现时必须完成以下四个关键任务。

（1）初始化软件环境，在本书的例子中就是初始化C语言的运行环境。

（2）将系统硬件和软件状态机初始化为确知的安全启动状态。

（3）对发生的各种中断立即做出响应，并且以相同的方式将它们提供的信息发送给系统的其他部分。

（4）工作于事件处理循环方式，不断检查各种系统事件，一旦发生某种事件就立即做出相应的处理。

尽管框架的各种要素因应用的不同而有所不同（特别是系统的初始化以及特定的中断服务处理），但是通用框架是构建应用系统的坚实基础。利用框架，我们能提高生产效率（由于框架都已经过测试），并且能将更多的创造力放在应用设计上（这正是我们创造新价值的地方），不必在底层问题上花费太多精力。

应用框架的主要问题之一是代码的结构，特别要保证代码能方便地根据特殊应用的需要而改变，同时又能保持其模块性，以便将其与系统的其他部分隔离。采用模块化代码后，由于对软件一部分的修改不会影响软件的其他部分，即使修改的地方有问题，也可以相对容易地测试并定位出错误代码，因而能够提高系统软件的稳定性。本书中的代码都是分组管理的，每组中包括完成相似功能的代码（如滤波或数据分析），某个C源程序使用的数据，以及其对应的头文件（里面包含需要使用例程的定义和函数声明）。这使得设计师能够将系统视为

142

① 1ksps = 1000次采样/秒。

一组功能部件的集合而不是一个“黑盒子”[①]（这种情况下，可能只知道代码是负责数据分析的，但是完全不知道也不关心它到底是如何进行数据滤波的），这些功能部件通过函数调用和共享全局数据实现相互通信；而分析程序只要简单地处理滤波后的数据即可。

程序框架包括以下10个主要模块。

(1) `Main.c/SystemEventDef.h`

内含应用程序的入口函数main()，它会调用系统初始化程序，然后启动事件处理循环。

(2) `SystemCfg.c/SystemCfgDef.h`

系统高级初始化程序，它们会调用针对ADC、滤波、数据分析以及通信模块的低级初始化程序，并完成独立的系统任务，比如读/写系统配置数据以及复位看门狗定时器等。

(3) `ADCIF.c/ADCIFDef.h`

包括采样传感器数据所用的模-数转换接口的初始化、处理以及中断服务程序。

(4) `Filter.c/FilterDef.h`

包括滤波器初始化以及高级数据滤波程序。

(5) `Analysis.c/AnalysisDef.h`

包括对滤波后的传感器数据进行处理以提取所需信息的程序和通过通信信道报告数据分析结果的程序。

(6) `CommIF.c/CommIFDef.h`

包括用于初始化特殊应用的信道以及控制通过信道的数据流的高级封装程序。

(7) `Timer.c/TimerDef.h`

包括定时器初始化和中断服务程序，用于执行时间紧迫型处理任务、调度数据采样以及记录消耗的系统时间。

143

(8) `Protocol.c/Protocol.c`

指CommIF.c中使用的介于应用系统与通信程序之间的高级协议处理程序。

(9) `Sensor.h/SensorDef.h`

包括对所采样数据的进行存储、检索以及应用传感器配置信息的程序。

(10) `dsPICDEMIF.c/dsPICDEMIF.h`

指为便于访问dsPICDEM1.1通用开发板上的硬件资源所需的封装程序。

当实现某个特定的应用时，上述大多数模块都需要针对特定的硬件平台以及实现应用所需的特殊功能而进行修改。但是，在这种代码结构中，可以以一种递增的方式进行上述修改，测试全部修改的代码。例如，在更换新的硬件平台后，首先需要修改的可能就是系统初始化代码。设计师可以先只执行初始化代码，而将其他代码通通屏蔽掉，待初始化代码全部测试完毕后再调试其他代码。一旦代码开始运行，ADC接口就会被激活，而其运行结果要等整个应用构建完毕并开始运行后才能检验。

如果不同版本的硬件平台发生变化，那么只需要修改`SystemCfg.c`、`ADCIF.c`和`dsPICDEMIF.c`模块中的代码，另一方面，如果我们只想从数据中提取不同的参数值，那么就只需修改`Analysis.c`模块中的代码。模块化对象集合了代码及其相关数据组，允许设计师复用软件中的大部分内容，从而能降低开发时间和成本，并且提高系统的可靠性。这三

① 对于“黑盒”系统，只要给定输入信号，就能推测出其输出，但是并不知道（也不必在意）系统的内部操作。很多人都熟悉的一种黑盒系统的例子可能是汽车的点火系统。我们都知道，如果旋转点火开关，那么汽车就会启动。但是汽车是如何启动起来的，通常对我们来说是很神秘而且也不必考虑的事。

个优点还增加了成品的生产率，并且简化了设计师的工作，从而实现双赢！

1. 初始化软件环境

任何用C语言编写的程序，都必须在程序启动后立即配置C语言运行时环境，然后才能执行`main()`函数中的应用代码。这部分初始化会配置很多重要的工作参数，比如栈和堆[①]的位置，设置编译器产生的各个数据段的变量初值等，最后，才调用`main()`函数启动应用

144

程序。表5-1中列出了可能使用的数据段以及存储在这些数据段中的数据类型。

表5-1 C30编译器产生的代码段和数据段

段名称	描述
text	通常并不是“数据段”，而是包含了可执行代码的代码段
data	初始化数据段之一，包含了设置全局变量初值的信息。.data段处理那些具有远属性的初始化变量，并且是采用大存储器模式编译的应用程序存放初始化数据的默认段
ndata	.ndata是第二个初始化数据段，用于处理具有近属性的初始化变量，并且是采用小存储器模式编译的应用程序存放初始化数据的默认段
const	.const段存放常值数据，比如文本字符串或者常量的数值。通常该段应位于程序存储器，并且可通过PSV窗口访问
dconst	所存放的内容与.const段类似，它是采用大存储器模式编译的应用程序在特殊条件下编译产生的
nconst	所存放的内容也与.const段类似，它是采用小存储器模式编译的应用程序在特殊条件下编译产生的
bss	第一个存放非初始化数据（未初始化的全局变量）的段。.bss段包含具有远属性的初始化变量，并且是采用大存储器模式编译的应用程序存放未初始化数据的默认段。在C程序环境初始化时，本段会被清零
nbss	第二个存放非初始化数据的段。.nbss段包含具有近属性的初始化变量，并且是采用小存储器模式编译的应用程序存放未初始化数据的默认段。和.bss段一样，在C程序环境初始化时，本段会被清零
pbss	第三个存放非初始化数据的段。.pbss段包含存放在RAM中的变量，它们的值不受器件复位（即dsPIC系列DSC芯片的复位）的影响。和存放在.bss和.nbss段的数据不同，.pbss段中的数据在初始化时不会被清零或者置为其他值。.pbss段位于近数据存储器

通常，C语言运行时环境初始化代码或者启动代码对应用程序设计师来说是透明的，它们都集成在C30编译器库文件`libpic30.a`中的`crt0.o`软件模块中。如果出于某种原因设计师不希望对已初始化的数据段（`.data`、`.ndata`、`.const`、`.dconst`和`.nconst`段）重复初始化，那么可以链接`crt1.o`文件而不是`crt0.o`文件；这两个文件的唯一差别是，`crt1.o`中没有数据初始化部分。此外，如果设计师需要在应用程序启动之前进行任何其他的初始化，那么它可以修改这两个启动模块中任何一个的汇编语言文件（`crt0.s`和

145

`crt1.s`，位于C30编译器安装目录下的`src\pic30`子目录中），然后对其重新编译即可。

必须注意的是，编译器提供的`crt0.s`和`crt1.s`源文件是针对dsPIC30F2010编写的，这可以从引言注释下面的头两行代码中看出：

```
...
        .equ_30F2010,1
```

① 栈和堆是子程序开始运行时建立的两种存储区。栈是一个存储器块，主要用于存放临时变量（也就是函数使用的局部变量）、函数传递参数以及程序返回地址等。在应用程序运行时，栈可能会增长或者收缩。

```
    .include"p30f2010.inc"
...
```

为确保代码通用于dsPIC系列的各种器件，Microchip C30采用了处理器指定包含文件，它们针对每个处理器定义了寄存器地址和位地址。程序员要确认所选择的包含文件适用于所使用的处理器，每个包含文件在最开始都有一条汇编指令，以检查是否定义了相应的器件ID，如果没定义就会报错。在上面的例子中，第一个语句定义了器件ID，第二个语句包含了该处理器对应的头文件。

如果应用系统使用的不是dsPIC30F2010处理器，那么程序员就必须向编译器指定正确的器件，以便生成代码。根据设计师的喜好，可以由以下三种方式中的任何一种来完成。

(1) 如果程序员使用命令行式的编译器，那么可以在调用编译器时使用命令行开关，例如：

```
C:\>AS-p30F6014A
```

(2) 在包含文件前使用.equ指示，例如：

```
.equ __30F6014A,1
.include"p30f6014a.inc"
```

(3) 选择MPLAB的菜单 "Configure→Select Device..."，在弹出的对话框中选择处理器，参见图5-3。(这也是最简单的方法。)

C程序初始化的最后一步是调用main()函数启动用户应用代码，下面就将介绍这些应用代码。

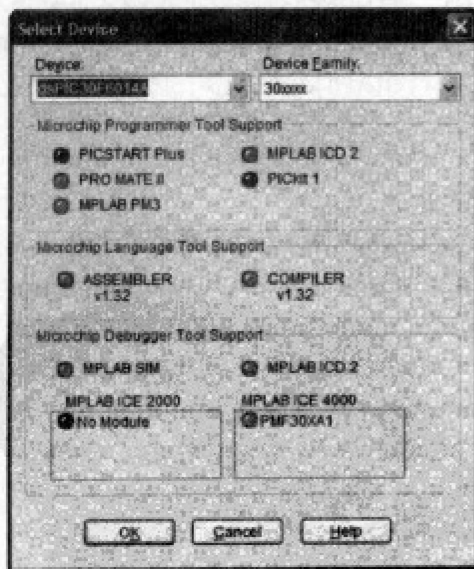


图5-3 MPLAB的Configure→Select Device对话框

2. 初始化系统的硬件和软件状态机

启动代码负责完成为建立软件环境以及软件运行（但不做任何与应用相关的配置）所需的最小初始化工作。这使得程序员能够灵活地配置C语言运行环境，并且与平台尽可能无关，但是，这也意味着程序员必须自己配置dsPIC系列DSC的I/O端口及其他片上外围设备。通常，会有一个简短的初始化周期，这段时间内I/O信号保持在复位时的状态，这可能不符合应用的要求。根据每种信号的用途，必须实现专门的硬件电路以保证系统从复位到完成软件初始化这段时间内能够安全运行。例如，如果某个信号是用于打开高功率激光的，那么我们可能不希望它在系统刚启动时就发射，因为即使是很短的激活时间都可能导致严重的损害。因此，应用程序代码必须将I/O端口初始化为确知的安全状态，这一点至关重要！

146

当然，对于大多数系统中，I/O端口配置只是整个系统初始化过程中的一小部分。最好的初始化顺序是先初始化I/O端口，然后按照时间紧迫性顺序将整个系统（不只是dsPIC系列DSC）的资源配置成期望的状态。根据不同的应用环境，可能需要在启动某些片上模块前，先初始化一些片外的部件，但关键是要按照部件要求的顺序尽快完成系统配置。设计师还必

147

须保证直到所有部件都被成功初始化后，才能开始处理系统事件。如果忽视了这个极为关键的准则就会导致系统错误，甚至是灾难性的故障。

有些设计师喜欢先初始化所有功能单元的硬件，然后再初始化与这些硬件有关的软件状态机，还有些人是在配置了相关硬件后立即初始化软件状态机。实际上，应当按照应用系统本身的特点来决定该采取那种方式。例如，有些系统就要求尽快将所有硬件模块都设置为期望的状态。如果可能，最好能够同时初始化某个功能部件的软件和硬件，因为这会使得代码更加模块化，并且有助于保证整个系统的部件都能正常工作。在本书的例子中，框架将采用模块初始化方法，以便于编程并且能够清晰地表达我们讨论的内容。

3. dsPIC的中断配置

配置中断的过程可分为四步。

（1）确认需要响应的中断源。

（2）为每个中断设置中断优先级（IPL）。

（3）使能每个中断，以便能被控制器识别。

（4）使能全局中断，以便dsPIC系列DSC能够处理所有已被激活和使能的中断源。

尽管上述几步对我们来说都不难，但是必须保证其中的3步能正确执行，否则系统就无法可靠工作。最坏的情况是有一个或者多个步骤部分执行正确，那么就会导致比较严重的系统错误，而且这些错误也很难确认、复现和解决。

本书的应用实例将使用三个相同的中断源，包括系统定时器、ADC以及信道。其中，系统定时器和ADC最为重要，因为它们决定了通过系统的数据流的顺序。信道尽管也很重要，但是由于它的功能不像另外两个模块需要极精确的时间窗口，因此它的优先级可略低一些。此外，如果得临时开辟缓冲区以接收，那么也不是什么大问题，但另一方面，如果系统是非周期性地启动采样，那么其性能就会迅速下降。

148

以下是按照优先级排列的应用系统需要响应的中断源。

- 定时器2/3，32位定时器模式，可变速率，用作ADC采样定时器。
- 定时器1，16位定时器模式，中断周期10ms，用作系统定时器。
- ADC采样就绪，当完成数据采样准备开始滤波时报告。
- UART1 Rx数据就绪，通信信道上接收到待处理数据。
- UART1 Tx保持寄存器空，通信信道上允许发送数据。

5.3 框架模块的实现

前面已经比较详细地讨论了启动代码，下面就看看程序框架是如何处理应用程序的唯一入口（只能是`main()`函数）的。5.2节已经提过，`main()`函数首先初始化系统部件，然后启动事件处理循环，在该循环里连续检查是否发生中断或者时间循环处理自身产生的事件。这些系统事件将被映射到16位全局变量`g_vui16SysEvent`中[①]，该变量定义在头文件`SystemEventDef.h`中。如果发现标志位为“1”，表明已发生事件，等待处理，如果它们被清零则表明事件已被处理完毕。

下面`main()`函数中的代码调用系统初始化程序：

① 变量`g_vui16SysEvent`显示出代码中的变量命名规则。通常，变量名前会加变量类型的缩写，比如，vui16表示易变的无符号16位数。全局变量则还会加g_以表示它们是全局的（而不是局部的）变量。

```
SystemInit():

Int16
  main(void)
  {
  //  Local Variables

  Uint8
    ui8Analysiscount,       // Decimation count for scheduling data analysis
    ui8RxData,              // Received communication data
    ui8Status;              // Function execution status

  //  Initialize the system hardware and the
  //  associated software state machines

  SystemInit();
  //  Process system events as they occur

  while (FOREVER)
    {
    //  Are there any pending system events? Start
    //  by checking for sampled data that is ready
    //  to be filtered

    if (g_vui16SysEvent)
      {
      //  Yes, is sampled data ready to be filtered?

      if (g_vui16SysEvent & EVT_FILTER)
        {
        //  Yes, data samples are ready so clear
        //  the event and filter the samples

        g_vui16SysEvent &= ~EVT_FILTER;
        ui8Status        = FilterData();

        g_vui16SysEvent |= EVT_ANALYZE; // Yes, signal
                                        //   Analyze Data
                                        //   event
        }

      //  Is there filtered data to be analyzed?

      if (g_vui16SysEvent & EVT_ANALYZE)
        {
        //  Yes, so clear the event and
        //  perform the analysis

        g_vui16SysEvent &= ~EVT_ANALYZE;
        ui8Status        = AnalyzeData();
```

149

```
//  Did the analysis reveal a condition
//  that should be reported?

ui8AnalysisCount++;      // Increment the data analysis decimation count

if (ui8AnalysisCount >= ANALYSIS_TIME)
    {
    ui8AnalysisCount = 0;                   // Reset the decimation count
    g_vui16SysEvent |= EVT_REPT_RESULTS;  // Flag that we have results
                                            //    to report
        }


//  Are there analysis results to
//  be reported to the host?

if (g_vui16SysEvent & EVT_REPT_RESULTS)
  {
  //  Yes, so clear the event and report
  //  the results via the communication
  //  channel

  FormatResultsMsg(g_ui8ResultsMsg,
                   &g_ui16ResultsMsgLength);

  if (g_ui16ResultsMsgLength <= CommGetTxFreeCount())
    {
    //  Have room in the transmit queue so add
    //  in the results message and clear the
    //  event to show that we've processed it

    CommPutBuff(g_ui8ResultsMsg,
                g_ui16ResultsMsgLength);
    g_vui16SysEvent &= ~EVT_REPT_RESULTS;
    }
  }

//  Has a 100 msec timer tick occurred?

if (g_vui16SysEvent & EVT_TIMER)
  {
  //  Yes, clear the event and perform any
  //  required timer tick processing

  g_vui16SysEvent &= ~EVT_TIMER;

  //  Insert any additional processing
  //  to be performed here
```

```
    }

// Check whether we have received
// any data from the host

if (CommGetRxPendingCount() > 0)
  {
  // Yes we do, so get the next received
  // character and process it
     ui8Status = CommGetRxChar(&ui8RxData);

     if (ui8Status == ST_OK)
         CommProcRxChar(ui8RxData, &g_ui8CommParseState);
     }

   // Reset the watchdog timer
   // to keep it from expiring

   ResetWatchdogTimer();
   }

 // We should NEVER get here if the system
 // is operating normally. The routine will
 // exit and the startup code will reset the
 // device if we get to this point.

 return ST_SYSTEM_FAIL;
 }
```

在系统初始化时需要注意两点：一个是必须首先分配局部变量，另一个是初始化的细节应封装在一个单独的函数里，而不要直接放在main()函数中。第一点保证能尽快将硬件初始化为确知的安全状态，第二点则是编写优秀的、模块化代码的范例，它们应该是与平台无关的（至少应该是高级别的）。

在深入研究系统初始化代码前，我们先继续研究main()函数。在系统初始化完毕后，代码将启动事件处理循环，它将连续检查并处理系统事件、检查并处理由通信信道接收到的任何数据、复位看门狗等。该循环将反复执行，除非处理器被复位。

实现时，该循环将检查并处理单次循环中出现的多个事件。根据处理事件以及检查并处理接收到的数据所需的最长时间，该循环的结构可能得修改为每轮循环只处理一个事件。这只需通过将if()语句改为if()…else if()语句即可。有一个必须满足的准则是，必须在看门狗定时器溢出前对其复位，否则整个器件就会被看门狗复位。通常，我们只希望在软件进入明显无效的处理状态时才被看门狗复位，但是决不希望因为处理时间稍长而无法完成整个主处理循环而被复位。

150～152

为了更好地理解我们是如何进行系统初始化的，下面将深入剖析SystemInit()函数。配置代码使用了dsPIC30F DSC Peripheral Library①中的函数，该函数库包含在C30编译器中，

① Microchip公司的dsPIC30F DSC Peripheral Library的产品编号为SW300021。

并可以从Microchip公司的网站上获得。该函数库的说明文档，请参阅位于编译器安装位置的MPLAB C30\DOCs目录下的16-Bit Language Tools Libraries文档①，该文档也可以从Microchip公司的网站上获得。这些函数会被封装到特定应用的代码中，因此，当应用系统的底层硬件或软件平台发生变化时（比如，可能有一个更好的函数库可以实现相同的功能），只需更换新代码即可升级固件。

与本书设计的应用系统有关的另一个重要方面是，利用Microchip DSP Library例程可以实现先进的数字滤波功能，在大多数情况下，我们都希望使用FIR滤波器。为了实现这种滤波功能，应用程序必须首先为每个滤波器建立并初始化一个数据结构。这种被称为`FIRStruct`的结构（其代码可以参见编译器提供的`dsp.h`文件）包含了为了维护滤波器抽头延时线以及设置滤波器系数所需的所有信息。创建这种结构十分简单，只需分配滤波器系数向量（并且将其初始化为所需的系数值）和抽头延时线向量，然后将该FIR结构声明为全局变量或者是在函数存在范围内始终有效的局部变量即可。作为一条规则，作者更喜欢采用第一种方法（声名为全局变量），因为即使应用代码的执行路径发生变化，滤波器也始终有效。但是，如果只将滤波器声明为局部变量，一旦新版本软件的执行路径发生变化，滤波器的数据结构就可能超出作用范围，从而可能被其他数据覆盖，那么再次使用该滤波器结构时就会导致错误的结果。

dsPIC Filter Design软件最重要的一方面是，只要单击几个按钮就能够建立一个带有滤波器结构及其相关的抽头延时线和滤波器系数的汇编语言代码。例如，为了给热电偶应用设计一个滤波器，我们只需在Filter Design软件中进行如下几步即可。

153

（1）在主菜单中选择 Design → FIR Window Design...，参见图5-4。

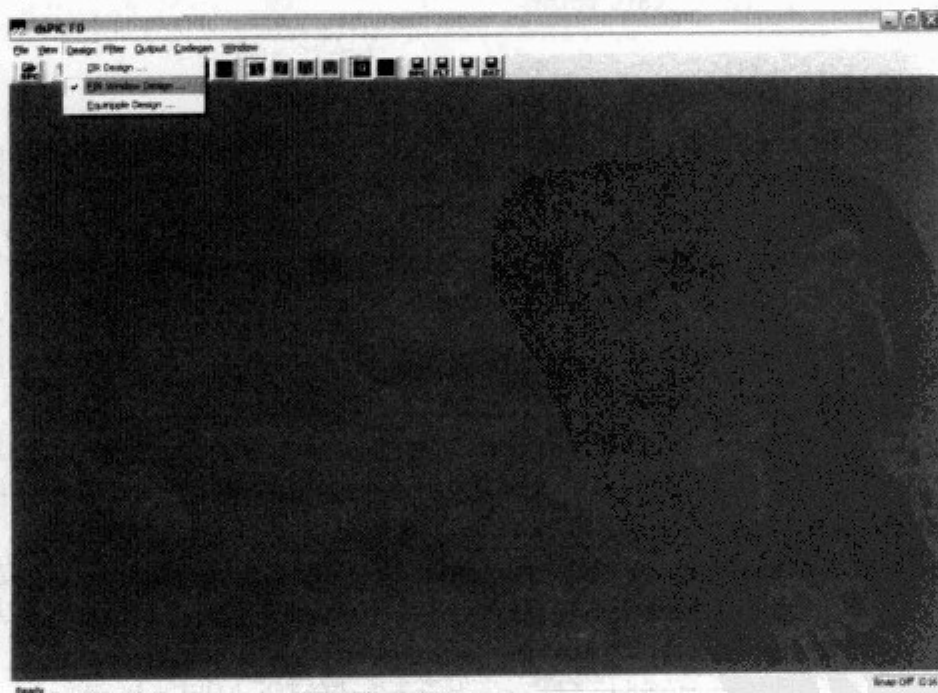


图5-4　选择FIR滤波器设计菜单

① Microchip公司的16-Bit Language Tools Libraries手册的文档编号为DS51456C。

(2) 该软件就会显示第一个滤波器设计窗口，参见图5-5。选择Lowpass滤波器选项，然后单击Next按钮。

(3) 输入图5-6所示的滤波器参数，然后单击Next按钮，进入第三个设计窗口。

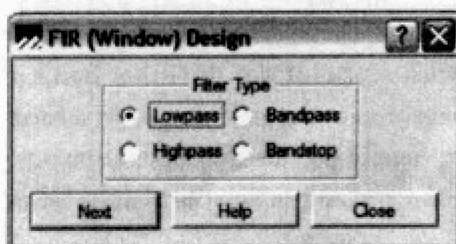


图5-5 FIR滤波器设计窗口

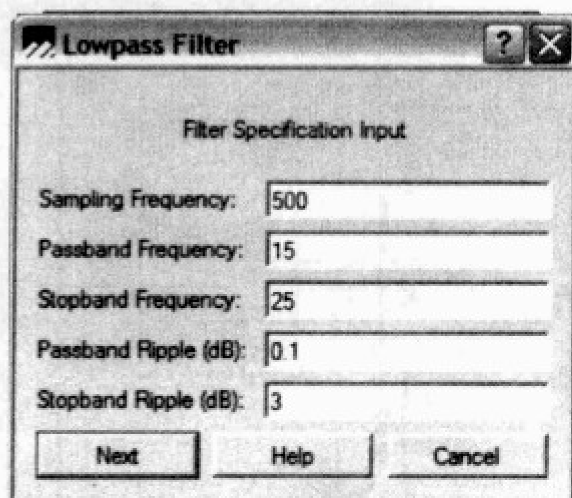


图5-6 FIR滤波器设计窗口

(4) 选择想要使用的滤波器类型，注意不同类型的滤波器为实现前面窗口指定的滤波器所需的抽头数。在图5-7中，我们选用Gaussian滤波器，它需要51个抽头。请注意，如果需要，我们还可以指定期望的抽头数。 154

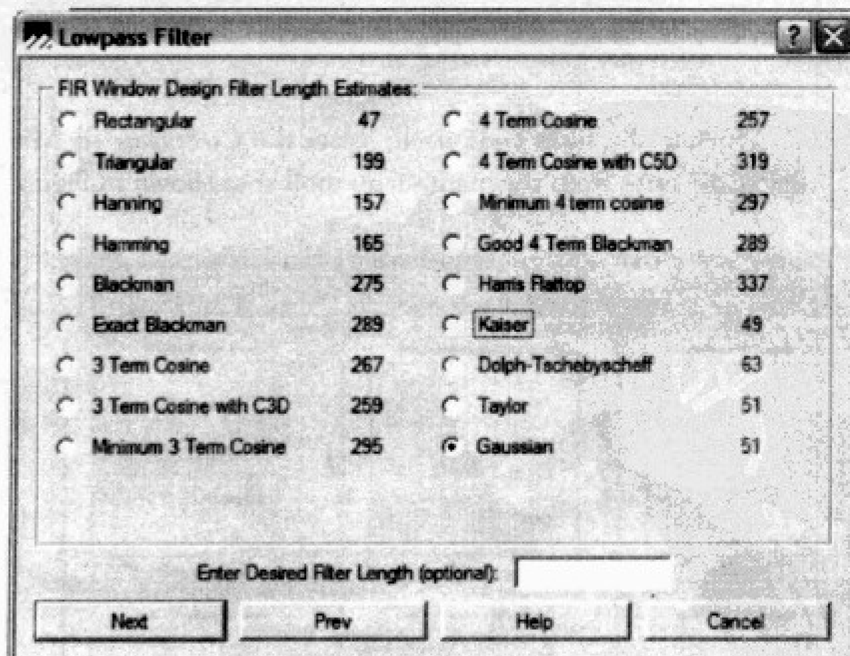


图5-7 FIR滤波器设计窗口

（5）在图5-7所示窗口中单击Next按钮后，软件会生成所设计的滤波器的响应曲线，参见图5-8。至此，我们就完成了滤波器设计，但是还没有生成代码。

155

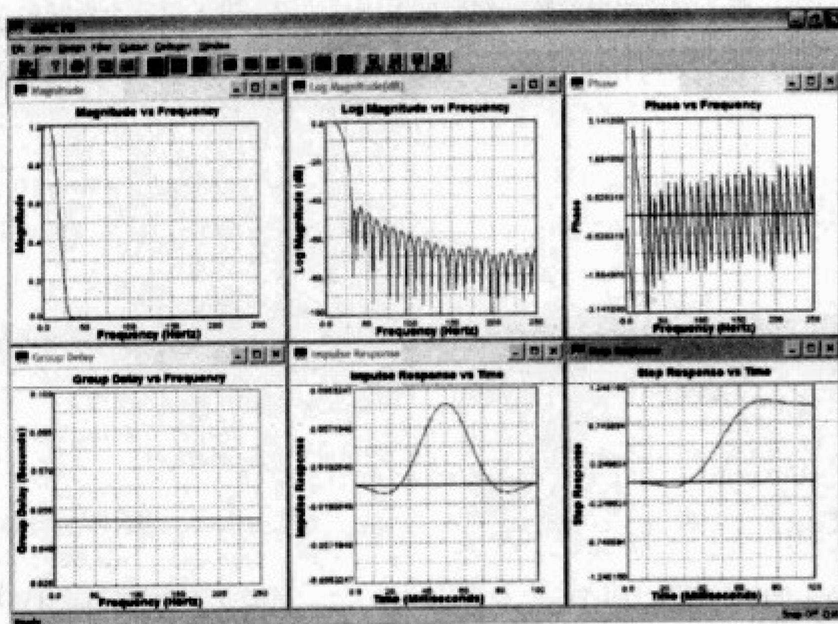
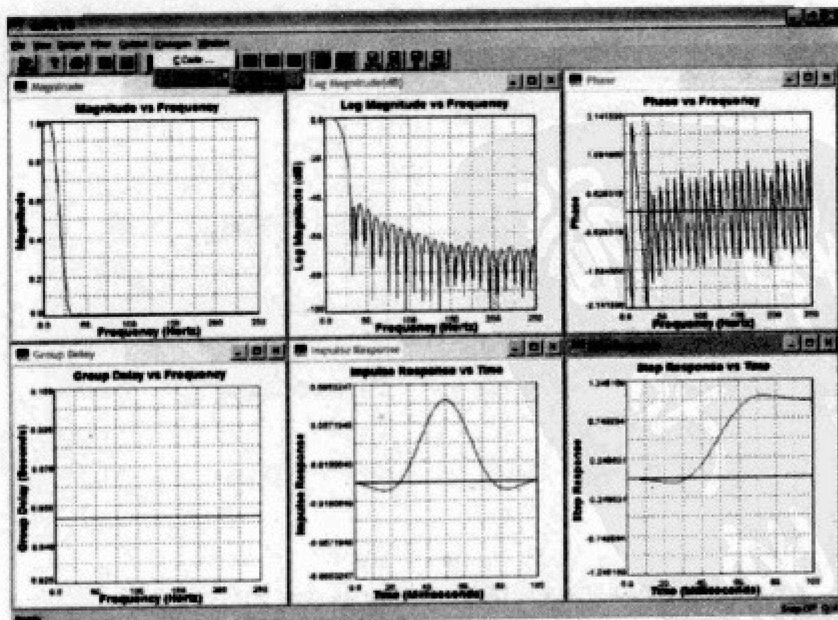


图5-8 FIR滤波器设计窗口

（6）为了生成滤波器的代码，选择主菜单CodeGen→Microchip→dsPIC30，见图5-9。



156

图5-9 选择FIR滤波器代码生成菜单

（7）软件会显示图5-10所示的对话框，要求用户输入所需的代码生成选项。这里使用默

认的选项（Use General Subroutine和X Data Space），并且再多选择一个C Header File and Sample Calling Sequence(.h)选项，它会告诉软件生成相关的C头文件。

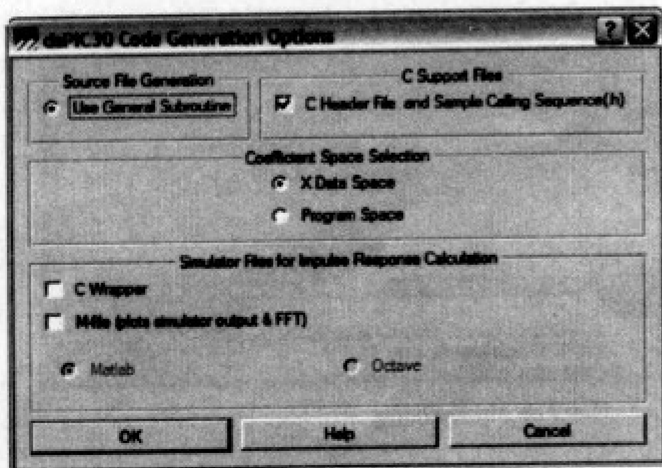


图5-10　代码生成选项对话框

（8）单击图5-10所示对话框中的OK按钮就会显示图5-11所示的dsPIC30 Code base file name对话框，它要求用户指定所生成的代码文件的名字。由于代码生成器将使用该文件名作为所创建的FIR滤波器结构体的名字，因此文件名不能有空格或标点。

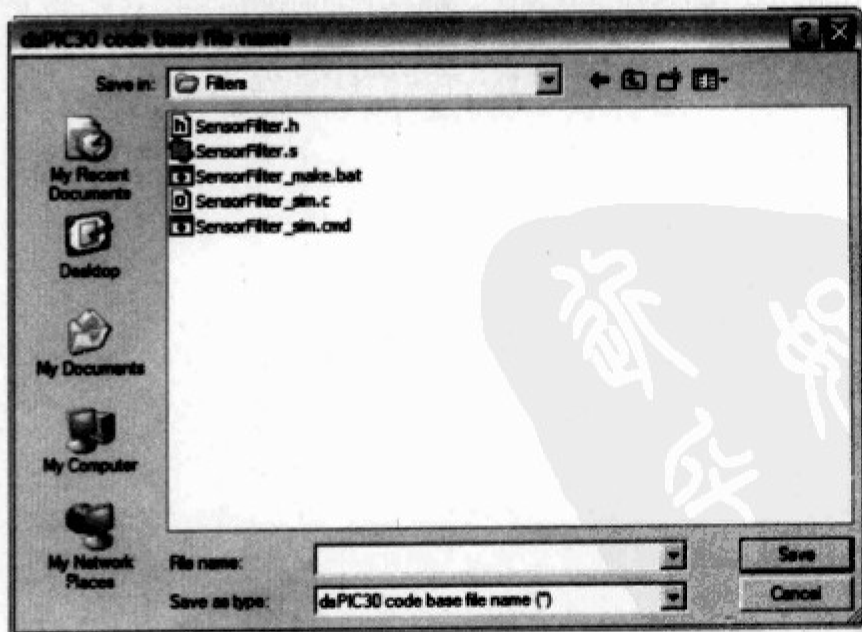


图5-11　dsPIC30 Code base file name对话框

输入文件名为SensorFilter后，单击OK按钮。

（9）软件会生成一组文件，对我们来说最重要的是`SensorFilter.h`（C头文件，其中定义了其他C文件需要使用的滤波器的结构体）和`SensorFilter.s`文件，这是包含滤

波器结构体及其相关系数和抽头延时线缓冲器的汇编语言代码。

为了在应用中实际使用所生成的滤波器，程序员只需将滤波器代码（这里就是SensorFilter.s文件）添加进项目代码。在应用程序第一次使用滤波器时，必须通过调用FIRDelayInit()函数完成初始化，它将初始化滤波器的抽头延时线为确知的状态。请注意，如果使用Filter Design软件包生成的代码，就不需要调用FIRStructInit()函数来初始化相关的结构体，因为该任务已由所生成的代码完成了。初始化抽头延时线失败将导致滤波器的响应出现错误，直到新数据完全通过延时线为止。 157

一旦滤波器初始化完毕，就能很容易地使用它处理新数据。当使用滤波器处理一组新数据时，应用程序只需调用FIR()函数，并指定进入滤波器的新数据的指针、保存滤波器结果的目标数据缓存的指针以及所用滤波器结构的指针。由于FIR()函数会自动更新相关的抽头延时线，因此应用程序无需考虑这项工作。

使用DSP库函数时需要注意的最后一点，也是最重要的一点是，如果库函数被中断，那么中断程序必须确保能恢复状态寄存器中的内容以及发生中断前DO和REPEAT指令的值，否则DSP函数将返回无效的处理结果。此外，DSP程序还使用了某些共享的硬件资源，如果中断服务程序改变了这些资源，那么这些变化将在中断服务程序执行完毕后导致部分DSP功能出现错误。为了方便设计师，每个DSP库函数都列出了它所使用的资源。 158

5.4 小结

本章设计出后续三章的应用系统设计实例中将使用的软件架构，这三个实例都是本章所讨论的理论的实际应用。尽管这里只是纲要性地介绍了软件框架的主要内容，但是我们很快就会为这些骨架添加新鲜的血肉，首先就是下一章将要研究的温度传感器。 159

第6章 传感器应用——温度传感器

在接下来的3章中（第6章至第8章），每章都会开发一套完整的传感器系统，它们能测量常见的物理量，并将测量值传输至主机。本章介绍温度传感器，第7章介绍一种压力和称重传感器，第8章介绍一个流量传感器。有必要说明的是，虽然每个应用只是针对某一类传感器，但是涉及的概念适用于很多敏感元件，并且通常只需稍做调整就能应用于某个用来测量我们所感兴趣的特定参数的专用传感器。读者可以大胆地拓展这里介绍的思想。一个很明显的例子是，同一个用于传感器参数测量的控制算法或许可用于各种场合并完成期望的效果。根据这个思路，下面我们将把注意力转向温度测量，它可能是现实世界中最常测量的物理量[1]。

温度的重要性在于，它对很多环境以及处理过程都有影响。我们，或者更为准确地说是我们所使用的设备，应能够精确地测量温度，以便调节房间内的温度、控制汽车发动机、烹煮食物、处理工业材料、监测病人的生命体征等。在第1章我们就介绍了一个温度测量仪器的例子——球泡型水银温度计，它能用温度计内部的水银高度指示绝对温度。但是，这种传感器很脆弱（通常是用一个玻璃柱包裹着水银）并且它还有毒性（水银是剧毒），从而限制了水银温度计只能在良好控制的环境中使用。本章开发的传感器系统将采用更能适应各种环境和应用的敏感元件：热电偶。在深入研究热电偶之前，首先了解一下可供选用的各种温度敏感元件。

6.1 温度传感器分类

近些年里，科学家研制出很多专用的敏感元件，它们能响应绝对温度，或者能随温度变化而改变自身的某些物理特性。本书主要关心可用电子器件监控的敏感元件，因为我们可以方便地将被检测物理量转换为可以用dsPIC系列DSC处理的电信号。这并不是说其他方法（比如通过视觉检测水银球泡型温度计）不可取，只是说那些非电子技术的检测方法不适合我们所使用的检测平台。这就好比铁钉和螺钉都能将两块板固定在一起，但是如果手边有一个榔头，那么显然使用铁钉更加方便。

目前，输出量可以用电子方式检测的敏感元件主要有5类。尽管以它们为基础还能构建出更为高级的敏感元件，但是这里只列出最基本的电子式温度传感器：

（1）热电偶；

（2）阻性温度检测器（RTD）；

（3）热敏电阻；

（4）硅温度传感器；

（5）红外温度传感器。

尽管接下来的几节会依次介绍上述传感器，但是本章开发的应用系统仅使用热电偶，理由是热电偶应用广泛、技术成熟、测量精确（使用得当的时候）并且相对便宜（这也是它被广泛应用的原因之一）。

① Temperature Technology Claimed to Eliminate Ambient Variations, *InTech* magazine, December 1998。

6.1.1　热电偶

热电偶是一种两线敏感元件，它能根据塞贝克效应测量两个金属连线结处的温度。塞贝克效应[①]是Thomas Seebeck于1821年发现的，它指出任何不同金属结上的电压会随结温而改变。尽管该电压变化量非常小，通常只有每度数毫伏，但是通过采用适当的模拟和数字信号处理技术就能基于热电偶构建出在很宽范围内都能精确测量温度的系统。

162

"采用适当的模拟和数字信号处理技术"是关键。为了能准确地测量出热电偶产生的微弱电压（毫伏级）信号，必须采用良好的接地和屏蔽设计，以防止测量电压中混入无法容忍的噪声。此外，由于印制电路板上引线的金属材料与制作热电偶的金属材料不同，因此用于测量原始热电偶电压的电路还会产生额外的塞贝克结，它们也会随温度而变化！最后，热电偶在其温度测量范围内具有很强的非线性，这可以从图6-1看出（该图是各种热电偶对温度变化的响应曲线）。

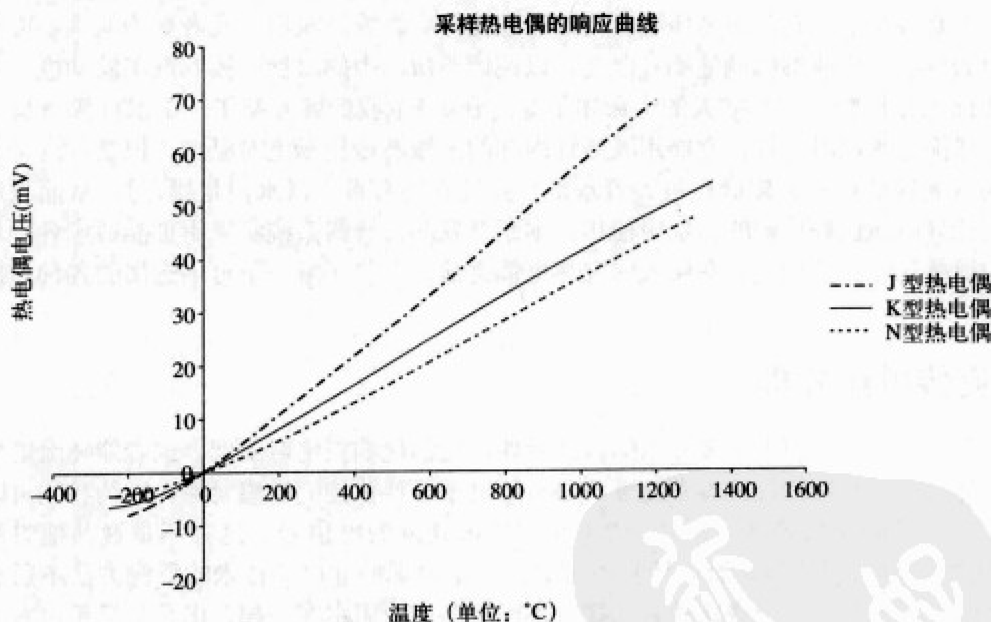


图6-1　各种热电偶的响应曲线

既然热电偶有这么多缺点，那为何还要选用热电偶测量温度呢？热电偶非常流行的原因主要有三点：第一它们的价格相对便宜，第二它们的温度测量范围较宽，第三其不足之处可以补偿（参见6.2.4节）。尽管为了克服热电偶的缺陷需要使用额外的电路和软件，但是它们能保证传感器的可靠性，并且其成本和性能满足很多应用的要求。

6.1.2　阻性温度检测器

另一种非常流行的温度传感器是阻性温度检测器（Resistance Temperature Detector，RTD）。它通常是由极细（直径很小）的铂金属线缠绕在不导电的圆柱或芯棒上，然后再用

163

① 介绍塞贝克效应的资料很多。比如*Electromagnetic & Electromechanical Machines, 2nd Edition*, Leander W. Matsch著，© 1977，Thomas Y. Crowell Company, Inc.。

绝缘材料将它们包裹起来而制成的。RTD器件的电阻会随温度变化而线性变化。只要给RTD器件通入已知大小的电流，就能根据其两端的输出电压计算出RTD的温度，计算公式为：

$$R_{RTD} = R_0 + \alpha T_{RTD}$$

其中，

R_{RTD}是RTD器件在温度为T_{RTD}时的电阻值，单位：欧姆。

R_0 是RTD器件在0°C时电阻值，单位：欧姆。

α是RTD器件的温度系数。

T_{RTD}是RTD器件的温度，单位：摄氏度（°C）。

RTD器件的温度系数取决于其所使用的铂金属线的纯度和成分。有两种标准RTD器件应用最广，一种的α值等于0.003 92，另一种的略小，等于0.003 85。由于这种α值更小的RTD器件最早在欧洲应用极广，因此也称之为*欧式曲线型*RTD，而另一种α值略大的则被称为*美式曲线型*RTD。

既然RTD器件的线性度比热电偶好得多，那么为何设计师过去总是选择热电偶而不用RTD器件呢？原因有两方面，一个是价格因素，另一个是系统性能的因素。由于RTD比热电偶更贵，因此对应的设备成本也更高。但是，通常更重要的原因是在系统性能方面。

通常，我们希望测量系统的温度时不影响被测系统本身。RTD需要一定的工作电流才能工作（这样才可能在器件两端产生电压），该电流会造成敏感元件自身发热。在某些应用中，要特别注意由于自发热而增加的热量会引起流过传感器的电流发生变化，从而导致随时间而变化的热误差。这种热误差是的的确确存在着的，它会使RTD器件周围的物体变得更热，从而无法准确反映出距离传感器很近的材料或者环境的温度。然而，热电偶在本质上就是零电流器件，它不会出现这种自发热现象。

另一个系统性能约束是RTD器件的温度测量范围，它比大多数热电偶的测量范围要窄得多（尽管个别类型的热电偶的测量范围可能与RTD的类似）。RTD的工作范围为–250～＋850°C，而热电偶的工作范围则达–270～＋2300°C。特别是在那些超过850°C的高温检测应用中，热电偶可能是唯一可用的敏感元件。 164

6.1.3 热敏电阻

和RTD类似，热敏电阻的阻值也会随温度的变化而改变。但是，RTD阻值会随温度的升高而增大，而热敏电阻的阻值则会随温度的升高而减小。这种*负温度系数*特性是区分这两种器件的唯一方法。与RTD不同的是，热敏电阻也具有极强的非线性。幸好可以通过适当地使用热敏电阻，利用其中一个非线性来补偿另一个的非线性，就能得到合理的线性输出。

由于热敏电阻也需要工作电流，因此它也存在和RTD一样的自发热问题，并且其工作范围比RTD或者热电偶都小得多，通常只有–40～＋150°C。尽管如此，通过合理配置热电阻仍能在该温度范围内实现极佳的精度①。

6.1.4 硅温度传感器

通常，在那些工作在极恶劣环境的电子系统中都没有温度传感器。系统可以利用所测得

① 有关RTD和热敏电阻的参考文献请查阅http://www.omega.com/techref/ measureguide.html以及http://rdfcorp.com/anotes/pa-r/pa-r_01.shtml。

的温度来启动空调或者在温度太高或太低而使系统无法安全工作时干脆关闭系统。系统还可以利用温度测量值完成其他应用任务。无论是何原因，这些年来随着系统器件集成度的不断提高，终于研制出基于硅片的温度传感器，并能将这种传感器集成到其他芯片中。

通常，硅温度传感器的测量精度极高（可以达到RTD或者热敏电阻的级别），但是它们的温度测量范围甚至比热敏电阻的还小（只有-40～+125℃）。由于任何硅温度传感器都需要能量才能工作，因此它也存在自发热问题。硅温度传感器比其他传感器的价格更高，但是由于它通常与检测电路集成在一起，因此无需连接线，使用方便。即使它们与处理单元有一定的距离，它们之间的连线也只需使用标准铜线即可，而无需像热电偶或者RTD那样使用昂贵的导线。

165

6.1.5 红外温度传感器

在有些温度测量应用中，不允许传感器与被测物发生物理接触，比如温度极高的物质（温度超出了热电偶2300℃的上限）或者在被接触后就会受损的材料，比如薄胶片或者未干的喷漆表面。在这种情况下，可以通过测量物体发射出的红外射线，再根据物体发射的电磁能量来计算出对应的温度。

尽管红外温度检测方法在理论上很简单，但是它在实现时面临诸多困难。最主要的困难是不同材料具有不同的*发射率*[①]特性即它们发射红外光的效率不同。如果希望合理使用红外传感器，就必须掌握被测物的发射率特性，否则温度读数就不准确。从实际例子中就能看到这种现象，比如在注模生产中监测金属桶内熔化塑料的温度。除非操作者知道他在做什么，否则测得的很可能是金属桶的温度而不是熔化塑料的温度[②]。

第二点可能是优点也可能是问题，这取决于是对一点还是一个区域进行红外温度测量。尽管对于所有传感器来说都存在这样的问题（毕竟，我们尚未发明出不占空间的传感器），然而我们前面已讨论过的其他传感器都是进行点测量，它们通常非常小因而可以忽略传感器自身的大小。而红外传感器则不同，它本质上是检测被测物上一个投影区域，因此报告的也是该区域内的平均温度。因此，在操作红外传感器时，必须完全覆盖被测物的检测区域，否则传感器就会报告不准确的读数（可能还包含被测物周围的温度）。

红外温度传感器第三个主要问题是它们会拾取来自被测物以及其他光源的反射红外线。像所有的电磁波一样，红外线会在它遇到的任何表面发生反射，因此传感器接收到的部分红外能量可能来自于其他红外源。如果传感器拾取的反射能量相对于传感器发射能量非常小，那么这种影响就可以忽略，但是如果反射能量所占比重很大，那么它会严重降低测量结果的准确性。

166

红外温度传感器的最大问题是它的成本。一个红外温度传感器往往售价高达数千美元，这是已经讨论过的任何传感器的好几倍。除了价格很高，它还要求对操作员进行额外的训练才能正确使用它，并且操作复杂（特别是要掌握被测物的红外辐射特性的详细信息），因此红外温度检测方法目前的应用范围有限。

---

① *发射率*通常表示为0%～100%或者是0～1之间的小数形式。无论是采用哪种表示，发射率越小表明红外光的发射效率越低。有关发射率的讨论请参阅http://www.omega.com/techref/measureguide.html。

② 在注模生产中，熔化的塑料通常被表示为熔体，而熔化塑料的前端被称为*熔体前沿*。

6.2 温度测量的要点

在任何温度检测系统中，为了得到有意义而准确的检测结果，就必须注意一些基本问题。如果对这些问题考虑不周，那么就会导致测量结果不准确。根据应用的具体情况，这些不准确性可能无关紧要（比如空调的恒温器坏了，那么你只需手动将温度调节到期望值即可），也可能导致失败（比如在实际温度只有250°F的烤箱内烘烤需要400°F才能烤熟的蛋糕），甚至可能引发灾难（比如氢气生产线的温度过高）。

在本章的其他部分，当讨论某个特殊问题时，尽管我们将着重讨论该问题是如何影响热电偶传感器的，但是所讨论的问题在任何温度测量系统设计中都应当引起注意，这和所使用的传感器类型无关。为了确保系统工作正常，我们将考虑以下几个重要问题：

(1) 可以采用的温度传感器类型；

(2) 要求的温度测量范围；

(3) 要求的分辨率和精度（这两个不等价）；

(4) 热电偶信号的特性；

(5) 已测信号中的噪声源。

6.2.1 测量范围

在任何测量系统中，我们必须首先确定需要处理的数据范围，因为这通常会决定我们能够使用的传感器。尽管热电偶能够测量–270～+1760°C范围内的温度，但是每种热电偶的测量范围只覆盖了其中的一部分，具体参见表6-1，这是摘自美国国家标准与技术研究院的热电电压和系数。[①]

表6-1 热电偶的温度测量范围

类 型	范围（°C）
B	0～1820
E	–270～1000
J	–210～1200
K	–270～1370
N	–270～1300
R	–50～1760
S	–50～1760
T	–270～400

然而，该表给出的只是每种热电偶使用的极限条件，还需掌握更多有关温度测量范围的知识。热电偶的实际工作范围还要根据其引线的直径（引线越细，温度测量范围越小）以及引线外皮（如果有的话）保护材料的工作温度范围而缩小。下面举例说明如何应对温度测量范围减小的影响。表中给出的J型热电偶的温度上限为1200°C，但是查阅商用型带保护裸线的J型热电偶的参考手册发现，由8 AWG（直径为0.128英寸[②]）导线制成的热电偶的测量范围上限只有760°C。如果热电偶是由36 AWG（直径为0.005英寸）导线制成，那么其最大测量温度就还会进一步下降到315°C，这比期望的温度范围下降了近75%。

① ITS-90的热电电压和系数表，可以从NIST的网址http://srdata.nist.gov/its90/ download/download.html下载得到。

② 1英寸相当于2.54cm。——编者注

实际中还有一些因素会影响热电偶的选择，因此设计师可能根本没有"选择"权。最终用户通常只有具有标准指标的热电偶，并且无法改变标准。比如在北美，很多注模生产线都使用J型热电偶，而欧洲和亚洲的客户在应用中又大都使用K型热电偶。明智的系统设计师应当使系统在任何情况下都能支持使用多种热电偶。这将节省最终用户和产品制造商的时间并减少很多麻烦。

6.2.2 测量分辨率

168 要求的测量范围不仅决定了我们需要使用的热电偶的类型，它还会影响测量的分辨率。分辨率是指能够测量的精细程度，通常用度来表示。对应用系统来说，能够分辨半摄氏度（°C）或者一个华氏度（°F）是很普通的要求。但是，只有在所使用的ADC以足够的分辨率对模拟信号进行数字化时才能满足上述温度测量的分辨率要求。为此，必须保证以下两个条件。

（1）必须将来自热电偶的模拟输出信号变换到能够被ADC数字化的电压范围内。

（2）ADC必须能够以期望的分辨率对变换后的模拟电压信号数字化。

尽管我们在前面大力赞扬了采用软件处理传感器信号的优势，但是上述第一个条件只能通过硬件（通常是由位于传感器输出到dsPIC ADC输入之间的放大和平移电路）来实现。如果我们无法将传感器输出的信号变换到适合ADC的电压范围，那么数字化结果就可能出现饱和[①]，当输入信号超出dsPIC系列DSC的数据手册指定的极限输入电压时还会损坏芯片。增加一级放大电路还能缓冲传感器的输出信号（它的驱动能量通常非常有限），使之适应dsPIC系列DSC的ADC输入端口相对较低的输入阻抗。此外，缓冲器还能防止dsPIC的ADC混入由于传感器输出信号过载而引起的干扰[②]。

然而，设计师该如何完成这种变换呢？其实该过程十分简单，只需要完成三个基本计算。但是在开始计算前，我们需要知道传感器输出信号的电压范围（至少要知道所测量的感兴趣量的大小），并且还要知道ADC允许的输入电压范围。具备这些知识后，就能进行以下计算了。

（1）计算传感器输出电压和ADC输入电压范围的跨度和偏置量：

$$SPAN_{SENSOR} = VS_{MAX} - VS_{MIN}$$
$$OFFSET_{SENSOR} = VS_{MIN}$$
$$SPAN_{ADC} = VA_{MAX} - VA_{MIN}$$
$$OFFSET_{ADC} = VA_{MIN}$$

169 其中，$SPAN_{SENSOR}$是传感器输出电压的跨度；

VS_{MAX}是传感器最大输出电压；

VS_{MIN}是传感器最小输出电压；

---

① 当理论的模拟输出电压超出实际可以达到的值时就会发生箝位，于是电路就无法传输期望的输出电压。例如某个放大电路的电源轨电压分别为地和+5V，标称增益为10。如果输入信号位于0～0.5V之间，那么该电路就能正确地放大它们。但是，如果输入信号大于0.5V，那么由于电路的最大输出电压只能达到（至少是接近）其供电电压，因此电路的实际输出电压只有5V。

② ADC电路使传感器输出过载的明显表现是，传感器的输出信号接示波器时看起来挺正常，可是一旦接入ADC输入端就会发生明显变化。这种变化可能表现为在传感器输入变化时输出的摆幅减小了，或者表现为传感器输出电压变为ADC的供电电压或者地。

$OFFSET_{SENSOR}$是传感器输出电压偏置；

$SPAN_{ADC}$是ADC输入电压的跨度；

VA_{MAX}是ADC输入电压的最大值；

VA_{MIN}是ADC输入电压的最小值；

$OFFSET_{ADC}$是ADC的输入电压偏置。

(2) 计算将传感器输出电压跨度映射到ADC输入电压跨度所需的放大倍数：

$$GAIN = SPAN_{ADC} / SPAN_{SENSOR}$$

其中，$GAIN$是所需的放大倍数。如果增益大于1，那么传感器输出信号就会被放大（即变得更大），如果增益小于1，那么传感器输出就会被衰减。

(3) 计算将传感器电压偏置平移到ADC所需的最小电压：

$$SHIFT = OFFSET_{ADC} - OFFSET_{SENSOR}$$

其中，$SHIFT$是在传感器输出电压放大后必须加入的偏置电压。

满足第一个条件（将传感器输出电压范围映射到适当的ADC输入电压范围）后，我们还需要进行以下检查，以满足第二个条件（验证ADC能够以足够的精度将变换后的电压范围内的信号数字化）。这也很简单，只需要3个步骤。

(1) 根据前面的计算得出所测参数值相对于传感器输出电压范围的变化范围：

$$PARM_{RANGE} = PARM_{MAX} - PARM_{MIN}$$

其中，$PARM_{RANGE}$是与传感器输出电压范围对应的参数变化范围，其单位由所测参数对应的物理量决定，而不是电压值；

170

$PARM_{MAX}$是与最大传感器输出电压对应的参数值（请注意这不是最大电压值，而是最大电压对应的参数值）；

$PARM_{MIN}$ 是与最小传感器输出电压对应的参数值（和$PARM_{MAX}$一样，它是参数值，而不是传感器输出电压值）。

(2) 计算ADC的分辨率，即确定ADC输入电压范围将被n比特分辨率的ADC分为多少等份：

$$RES_{ADC} = 2^{NumBits}$$

其中，RES_{ADC}是ADC的分辨率，它是以ADC的输入电压范围将被等分的份数或者量化等级的形式表示；

NumBits 是ADC进行数字化的位数（比如10或者12，这主要取决于dsPIC DSC的类型）。

(3) 计算所测参数的分辨率，并表示成单位ADC量化等级对应的参数值的形式：

$$PARM_{RES} = PARM_{RANGE} / RES_{ADC}$$

其中，$PARM_{RES}$是参数分辨率，表示成单个ADC数或等级对应的参数单位值。

下面将通过一个简单的例子来说明这些概念。假设要使用J型热电偶来测量0～750°C范围内的温度，并且假设已经选定了所需的热电偶。因此，这里只需设计信号处理电路。从ITS-90表中J型热电偶的热电电压可以看出，J型热电偶在0°C时输出0.0mV，而在750°C时输出42.481mV。为便于缩放，假设输入电压范围是0～50mV（近似对应870°C），并假设所使用的dsPIC ADC的输入电压范围是0～5V。因此放大电路所需的增益和偏置如下。

171

(1) 计算传感器输出电压和ADC输入电压的跨度和偏置：

$$SPAN_{SENSOR} = 0.050V - 0.000V = 0.050V$$

$$OFFSET_{SENSOR} = 0.000\text{V}$$
$$SPAN_{ADC} = 5\text{V} - 0.0\text{V} = 5\text{V}$$
$$OFFSET_{ADC} = 0.0\text{V}$$

（2）计算将传感器输出电压跨度映射到ADC输入电压跨度所需的放大倍数：

$$GAIN = 5\text{V} / 0.050\text{V} = 100$$

（3）计算将传感器最小输出电压映射到对应的ADC输入最小电压所需的电压平移量：

$$SHIFT = 0.0\text{V} - 0.000\text{V} = 0.0\text{V}$$

这里，由于模拟输入电压范围和模拟输出电压范围都有下限0V，因此模拟放大电路中无需电压偏置。然而在通常情况下，放大器电路都需要一个电平平移电路，以便将输入电压范围的下限映射到期望的输出电压范围的下限。

通过映射输入电压范围，我们可以验证出第二个条件也是成立的。如果dsPIC系列DSC采用12位ADC，那么ADC输出的1个数位就对应ADC输入电压范围的1/4096（$1/2^{12}=1/4096$）。对于我们的应用实例来说，这就意味着对放大后的温度信号的数字化分辨率可以达到：

$$T_{RES} = T_{RANGE} / RES_{ADC}$$
$$T_{RES} = (870^{\circ}\text{C}-0) / 4096$$
$$T_{RES} = 0.2124^{\circ}\text{C}$$

其中，T_{RES}是系统的温度分辨率，单位：°C；

T_{RANGE}是系统的温度输入范围，单位：°C；

RES_{ADC}是ADC的分辨率，单位：份数。

通过简单的运算就会发现，如果采用带有10位ADC的dsPIC系列DSC（其分辨率为 [172] 2^{10}=1024份），那么能达到的温度分辨率就会下降到：

$$T_{RES} = (870^{\circ}\text{C} - 0) / 1024$$
$$T_{RES} = 0.8496^{\circ}\text{C}$$

如果希望达到1°F（相对于0.56°C）的温度分辨率，那么就得使用12位ADC，10位ADC无法满足要求。

6.2.3 测量精度

用户经常会（有时设计师也会）混淆系统的分辨率与精度，它们是完全不同的概念。分辨率表示可用于计算测量值的粒度的大小，而精度则表示测量读数的正确程度。这两个概念都很重要，但是必须注意二者之间没有蕴含关系。例如，系统的分辨率可能达到0.5°F，但是，如果因为系统的故障而导致测量值都带有10°F的偏差，那么分辨率再高也没有用。

那么为何会出现这种异乎寻常的错误呢？第一种原因是传感器本身有问题，测量不准确的传感器就会引入不精确性，如果这种不精确性可以补偿，那么它还可以接受，否则就无法接受。实际上，不同类型的热电偶在其工作范围内的精度也不同，比如J型热电偶的最高精度可达± 0.1°C，E型、R型和T型热电偶最高精度可达± 0.5°C，K型热电偶的精度为± 0.7°C，而S型的只有± 1.0°C。

此外，信号链中的其他因素也会导致得到错误的结果，包括缺少必要的冷结补偿处理（该内容将在冷结补偿一节讨论），无法有效抑制共模噪声，或者电路中引入了非共模噪声等。通常，我们可以补偿由传感器或者电路其他元件自身引起的不精确性，由于这些问题不是时

间或者温度相关的，因此可以通过标定和线性化进行补偿。需要记住的关键问题是，分辨率不等价于精度，因此我们必须同时保证这两个指标才能构建出稳健而可靠的系统。

6.2.4 挑战

下面，我们将把注意力从普通问题转向特殊问题，从适用于所有传感器系统的系统设计方面转向使用热电偶作为敏感元件会遇到的特殊情况。

173

1. 信号特性

传感器系统设计的第一步就是要确定传感器输出信号在感兴趣范围内的特性。这些特性包括信号的电压大小、信号的输出驱动能力以及预计的信号频率成分。

a. 信号的大小

通过查阅NIST的热电压表格可知，热电偶产生的输出电压非常小，通常只有数毫伏。使用热电偶信号的困难之处在于它是一种零电流信号，这意味着它只能驱动极高阻抗的负载。但是，dsPIC的ADC的输入阻抗只有20kΩ，因此不满足这个要求。最后，根据我们需要测量的温度范围的不同，热电偶的输出电压还可能是负值，这也超出了dsPIC的ADC允许的输入信号电压范围。即使热电偶在所有我们感兴趣的温度范围内都输出正电压，但是热电偶的引脚连线也可能需要反接（实际中经常会发生这种情况），因而数字化电路的输入电压可能还是负的。显然，基于热电偶的温度测量系统需要在传感器输出和ADC输入之间采用高阻抗放大器实现信号的缓冲、放大和平移。

通过适当选择放大器的增益和偏置电压，我们就能设计出合适的放大电路，它在热电偶连线正确或者引脚反转的情况下都能正常工作。乍一看，这种能力可能看起来不是很重要，难道用户不能通过简单地交换热电偶的连线来解决问题吗？有时是允许这么做的，但是很多情况下不允许这样。在作者参与的一个注模生产项目中，移动模具（热电偶就安装在其中）和进行任何改动的最小成本是50 000美元，而机器每天运行的成本则为20 000美元。整个系统包含100多个不同的传感器，根据墨菲定律[①]可知，其中至少有一个热电偶的连线会出错（事实上的确如此）。在这种情况下，如果传感器具备在少量连线错误的条件下正常工作的能力，就可以为用户节省大笔资金和麻烦。

b. 频率成分

如果希望对传感器输入信号滤波以降低电气噪声，那么就需要确定传感器信号的频率成分以便掌握可以安全衰减的频段，同时又不过分地衰减信号本身。为了确定温度信号的频谱成分，我们需要掌握一些被测系统的热特性，以及温度变化的速度等。有些系统的温度可以迅速改变，而有些则变化相对较慢。尽管设计师往往事先并没有这些特性的准确指标，但是他可以很好地估计出温度变化的最大速率。比如，在加热系统中，根据加入元件的功率和被加热的媒质的热损耗特性就能估计出温度变化的最大速率。比如，当打开烤箱加热比萨时，我们不可能期望其温度会立即升高到400°F，实际上通常需要3分钟才能达到。利用这种公认的原始方法，我们就能估计出该系统的最大温度变化速度[②]约为：

174

$$\Delta T_{MAX} = 400^{\circ}F / 180s = 2.22\ ^{\circ}F/s$$

① 墨菲定律简明地指出当可能出错的时候就一定会出错。并且在给定的可能有问题的两个选择中，问题很可能出在那个更关键的选择中。

② 温度变化速度与标准的速度的概念类似，标准的速度是指单位时间内位移变化量，而温度变化速度则表示单位时间内的温度变化量。

如果继续以在测量分辨率一节中所使用的12位J型热电偶为例，并注意1°C=1.8°F，那么就可以计算出该最大温度变化速率对应的ADC量化单位个数为：

$$\Delta Count_{MAX} = \Delta T_{MAX} / T_{RES}$$

$$\Delta Count_{MAX} = (2.22°F/s) / (0.2124°C/次 \times (1.8°F/°C))$$

$$\Delta Count_{MAX} = 5.8次/s \approx 6次/s$$

假设要求跟踪1个ADC量化单位的温度变化，那么意味着每秒至少得采样6次。但是，根据第2章学习过的奈奎斯特判据可知，理论的最小采样速率应该至少是信号最高频率的两倍（即12次/秒），而实际上的采样速率大约是信号频率的5倍（30次/秒）。

采用这种方法时有各种注意事项。首先，最明显的一个是，实际的循环加热周期内的温度变化速率可能比利用该方法预测的值更大。此外，这里还假设了传感器信号经过适当的模拟抗混叠滤波器，其宽带噪声被抑制。如果实际情况并非如此，或者如果滤波器的带宽比所需的采样速率更大，那么就需要适当地增大采样速率。最后，热电偶本身也有响应时间特性，也就说热电偶需要一定时间才能使它的输出电压准确地反映节点温度。尽管如此，在缺少额外信息时，该方法可能是个较好的选择。

175

2. 冷结补偿

到目前为止，有关热电偶接口中最困难、也是最不了解的方面是所谓的*冷结补偿*（Cold-Junction Compensation，CJC）概念。回忆前面有关热电偶的讨论可以发现，塞贝克效应指出，由不同金属构成的结两端会产生与结温度有关的电压，热电偶就是根据该效应发挥作用的。但是，不仅热电偶的两个引脚间（即所谓的“热”结）存在塞贝克效应，而且热电偶的引线端部与电路板的铜导线连接处（即所谓的“冷”结）也存在塞贝克效应。我们必须去除这种不希望出现的塞贝克效应电压，因此，需要使用所谓的*冷结补偿*技术。

冷结补偿的基本思想[1]是测量冷结（即热电偶引线在PCB上的接入点）的温度，并且将该温度值加入热结（即作为传感器的热电偶）的温度计算中，从而补偿不希望的塞贝克效应电压。一种等效的方法是将冷结电压加入热结电压中。为了正确实现，设计时必须确保：

(1) 与冷结相连的热电偶引线两个端点必须保持相同的温度（即所谓的*隔热端点*或*隔热势垒*）；

(2) 用于测量隔热势垒温度或者产生对应电压的设备应尽可能靠近隔热势垒。

如果不满足上述两条要求，那么就可能导致温度测量出现不可修复的误差，严重时该误差还会随温度而改变。

3. 线性化

除了冷结补偿之外，处理热电偶信号时最困难的工作就是将非线性的热电偶输出电压映射到线性的温度尺度上。其中一种常见的方法是通过迭代法推算转换多项式，其中多项式的系数与热电偶的类型有关，但是由于该方法的运算量较大，因此通常不适合于嵌入式处理器应用。尽管dsPIC系列DSC能够高效地完成数学运算，但是还有一种*分段线性化法*，它可以通过更加简单的实现方法得到极佳的结果。

176

分段线性化法的基本思想是将非线性曲线分成若干个线性段，参见图6-2。在每段中，我们可以将实际曲线近似为连接本段起点和终点的直线段。为了计算估计值，我们可以首先确定包含待估算值的线性段，然后利用对应段的斜率和Y轴截距计算出估计值。这样就能用

① Circuit Provides Cold-Junction Compensation, Mark Maddox and John Wynne. *EDN*, November 11, 2002, http://www.edn.com/ article/CA260064.html.

搜索与线性计算来代替高阶多项式运算。

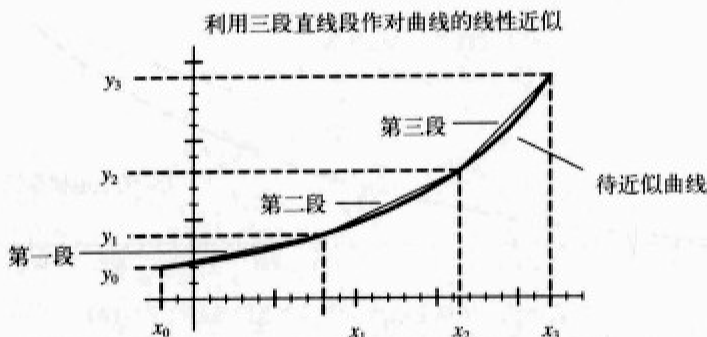


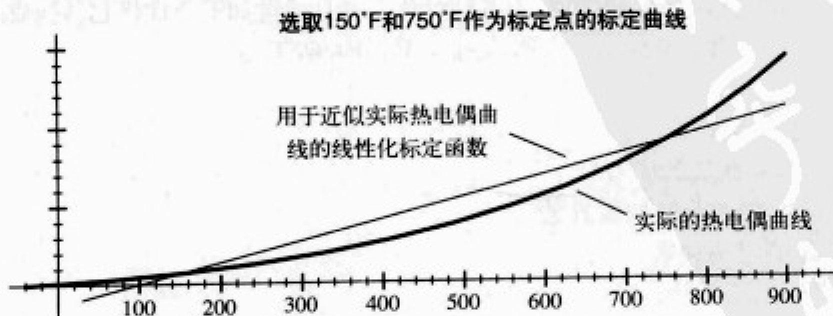
图6-2　曲线的分段线性化实例

影响该技术效率的因素有两点：一个是用于近似曲线的线性段的个数，另一个是待近似曲线本身的特性。显然，使用的直线段越多，那么与曲线的匹配度就越高，但是需要花费更多的时间以确定用于线性化的直线段。这种平衡作用使得设计师多了一种工具，可以降低精度来提高速度，也可以降低速度来提高精度。

4. 标定

尽管线性化是将温度电压信号转换为dsPIC硬件可处理的数字量过程中很重要的一部分，但是为了得到有用的数字量，还需要其他处理过程。必须利用一组已知的温度数据对线性化后的信号进行标定，才能确保硬件计算后的信号值足够精确。到目前为止，应用最广的标定热电偶的技术是两点法，它选取两个不同温度点处（通常取感兴趣的温度范围的上下限附近的某点）的测量值制成线性化标定参考曲线，然后实施标定。

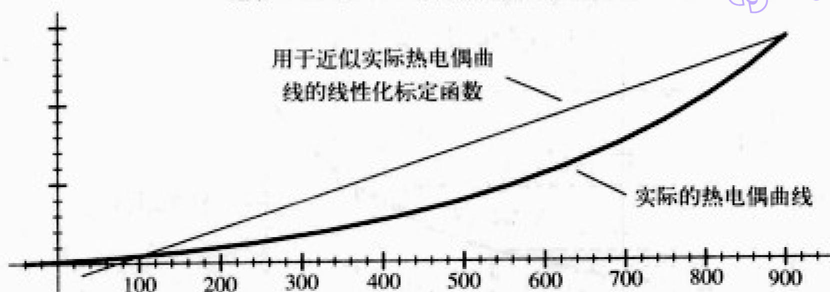
例如，假设希望测量100 ~ 900°F范围内的温度。那么可以取150°F和750°F作为标定点，因为它们都位于温度测量范围的端部附近，从而能保证我们生成的线性曲线与实际的热电偶曲线相差不大。尽管也可以选取100°F和900°F作为标定点，但是所得线性化曲线的误差会更大，具体参见图6-3a和图6-3b。



(a) 利用150°F和750°F作为标定点来标定曲线

图　6-3

选取100°F和900°F作为标定点的标定曲线



请注意，为了便于表示线性化标定函数与实际热电偶曲线之间的误差，图中的热电偶曲线的弯曲有些夸大。在选取这组标定点的情况下，两个标定点之间的误差明显增大，但是在接近温度测量上限（900°F附近）时逐渐减小

(b) 利用100°F和900°F作为标定点来标定曲线

178

图6-3 （续）

为了说明标定过程，我们将分别测量低于和高于标定点温度时的线性化电压，并计算标定曲线的增益：

$$G_{CAL} = (T_{UCP} - T_{LCP}) / (V_{UCP} - V_{LCP})$$

$$G_{CAL} = (750°F - 150°F) / (V_{UCP} - V_{LCP})$$

$$G_{CAL} = 600°F / (V_{UCP} - V_{LCP})$$

其中，偏置量为：

$$OFF_{CAL} = T_{LCP}$$

在上述公式中，

G_{CAL}是标定后参考曲线的增益；

OFF_{CAL}是标定后参考曲线的偏置量；

T_{UCP}是上标定点的温度（750°F）；

T_{LCP}是下标定点的温度（150°F）；

V_{UCP}是上标定点的电压读数；

V_{LCP}是下标定点的电压读数。

实际中，传感器固件会接收电压读数，对其线性化，并且通过如下公式将它转换成温度值：

$$T_{CAL} = G_{CAL} \times (V_{READING} - V_{LCP}) + OFF_{CAL}$$

其中：

T_{CAL}是标定后的温度读数；

G_{CAL}是标定后的参考曲线的增益；

OFF_{CAL}是标定后的参考曲线的偏置量；

$V_{READING}$是线性化的电压读数；

V_{LCP}是下标定点的电压读数。

上述讨论提出一个明显的问题：如果要标定温度传感器，那么怎么才能知道两个标定点的温度是多少呢？答案是我们必须使用一个已经标定过的设备作为参考。比如使用热电偶标定器，它能根据特定温度输出精确的电压。热电偶标定器的价格便宜，供应商也很多，但是更精确和易用的型号的价格则达数百至数千美元。

179

5. 噪声源

这里将介绍很多会干扰热电偶系统的噪声源。由于热电偶固有的信号电平较低，因此电气噪声的破坏性极大。幸运的是，热电偶的差分信号具有固有的噪声消除特性以及温度信号和噪声频谱，能有效地滤除大部分不需要的信号。

交流电源

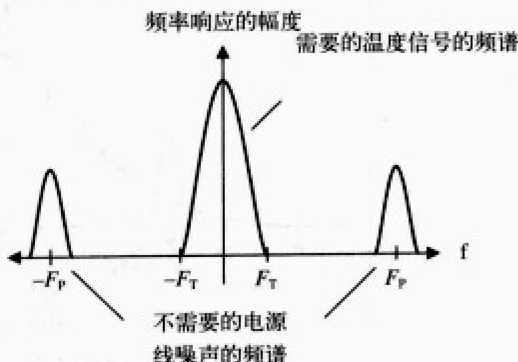
最常见的一种电气噪声源来自交流电源线辐射，特别是在工业或在热电偶附近存在通有大电流的导线的环境中，噪声更严重。电源线噪声具有一个可以弥补的性质：它的频谱带宽极窄，并且中心频率位于50Hz或者60Hz附近（不同国家有所不同），而谐波成分则是这些频率的整数倍。对于设计师来说，这种定义良好的窄带噪声信号实际上比低电平的宽带噪声更容易处理，这是因为窄带噪声信号的带宽只与传感器信号频带有少量重叠。

由于热电偶的输出信号太小，并且驱动电流几乎为零，因此很容易被电源线污染。但热电偶信号会给我们的设计带来两点益处：一个是差分信号有助于抑制共模噪声，另一个是电源线噪声的频谱通常位于温度信号的带宽之外。即使那些电源线噪声的频率位于温度信号的带宽之内，由于电源线噪声是窄带的，因此可以通过带阻滤波器有效地将其滤除。

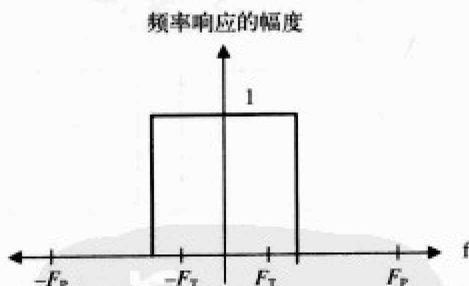
图6-4a展示了带外的电源线噪声的频率成分以及温度采样信号的频谱。利用图6-4b所示的低通滤波器，我们就能很容易地去除寄生频谱成分，从而得到图6-4c所示的滤波后信号。

图6-5a展示了更为困难的带内的电源线噪声的情况，此时电源线噪声位于温度信号频谱之内。因此，我们必须采用高阶的陷波器（其边沿很陡峭）去除无用的频率成分，留下更好的但是并不是很清晰的滤波后的温度信号。即使这样对信号略有衰减，但是结果还是比滤波前的信号更准确。

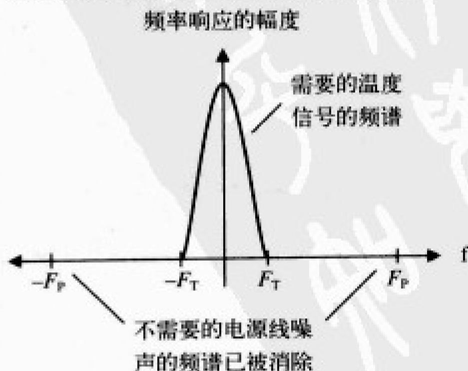
图6-6展示了电源线噪声频谱与温度信号频谱严重重叠的情况。这时，没有什么好的滤波方法可以补偿电源线噪声的影响，



(a) 采样含有带外电源线噪声的温度信号



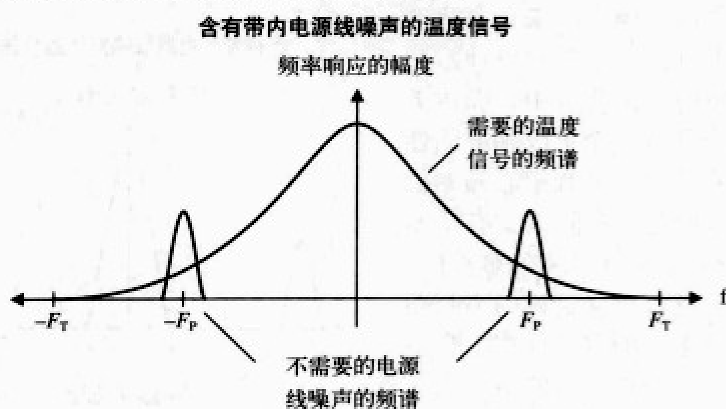
(b) 利用低通滤波器滤除带外电源线噪声



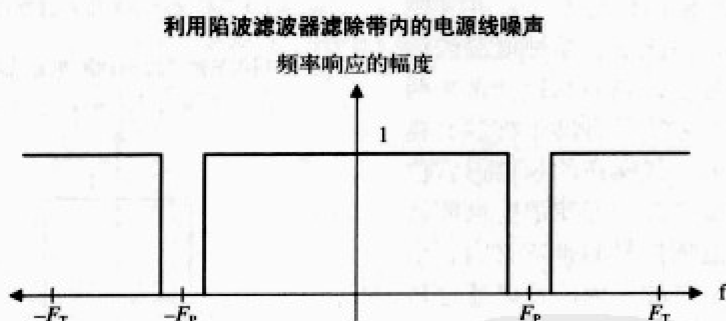
(c) 滤波后的传感器信号

图 6-4

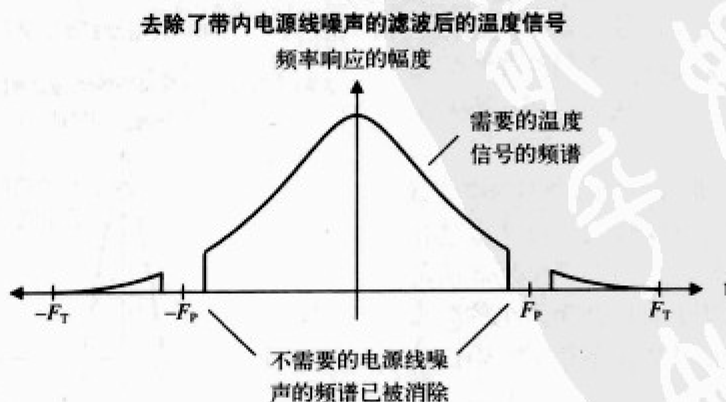
于是设计人员只能采用替代的方法，即比较热电偶上的电源线噪声与电源线信号本身，然后从所测得的传感器信号中将其减掉。尽管这种方法在技术上是可能实现的，但是更好的方法是将热电偶与电源线屏蔽，并且优化对电源线的布线，以减小耦合辐射。实际中，大多数基于热电偶的系统都不会碰到这种极端的情况，因为只有极少数材料（包括制造热电偶的材料）能响应以50Hz或60Hz速率变化的温度变化。



(a) 采样含有带内电源线噪声的温度信号



(b) 利用陷波滤波器滤除带内的电源线噪声



尽管滤波器也会影响需要的温度信号的频谱，但是大部分原始信号被完好地保留下来，因此我们仍然能够得到有用的结果

(c) 滤波后的传感器信号

图 6-5

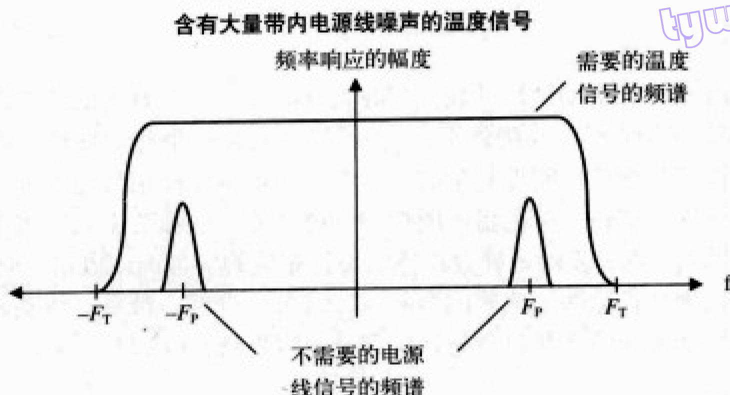


图6-6 温度信号与电源线信号的频谱重叠时的采样结果

6. 故障条件

在热电偶上端接高阻抗偏置电阻并且注意被测热电偶电压的特性，就能很容易地检测出以下两种严重的错误。如果无法检测出这两种错误，就会导致传感器可能会报告非常不准确的测量值，并且根据使用测量值的方式，还可能会导致灾难性的系统故障。

a. 热电偶开路

第一种可以检测出的错误是热电偶开路（即热电偶两引脚上的连接结已被切断）。这种情况经常出现在那些存在严重的机械冲击和热冲击（特别是反复被冲击）的系统中。通过在热电偶输入连线上添加两个匹配的高阻抗电阻就能直接检测出这种错误，具体参见图6-7。一旦结断开，热电偶的引脚就会被偏置到输入电压范围的极端处，这就很容易被dsPIC系列DSC检测出来。

182 ~ 183

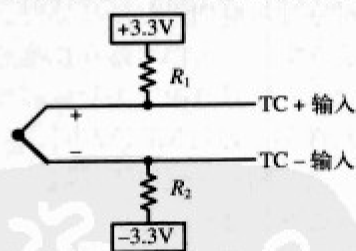


图6-7 在热电偶输入端添加敏感电阻

在选择偏置电阻的阻值时，设计人员需要保证正常情况下热电偶自身产生的信号不会被淹没。通常，选择数十万欧姆的电阻就可以，但是这两个电阻要尽可能匹配，以便保持热电偶信号的差分特性。

b. 热电偶反接

由于热电偶的引脚有极性，因此它们可能会被接反。尽管信号电压很小，这种错误不会损坏输入电路，但是如果无法检测出这种错误，就会得到错误的测量结果。当热电偶反接时，测量到的信号电压会随温度的增大而降低，但是由于我们通常不知道实际温度到底是增大了还是减小了，因此确定热电偶是否接反的唯一方法是检测信号是否超出正常输出范围的下限。例如，J型热电偶在低于0°C的温度下会产生负电压，因此，如果感兴趣的温度测量范围是0～500°C（对应32～932°F），那么一旦出现负电压就表明热电偶接反了。

由于我们希望能够在热电偶输出电压低于某个最小值时识别出它接反了，并且还希望能够发现热电偶开路，因此要求系统的数字化范围的上下限都要有一定的裕量。这样，我们就能确保测量出整个感兴趣的温度范围，并能正确地辨识出热电偶的错误。

184

6.3 应用设计

具备了前述的相关背景知识后，我们现在就开始设计一个真正的智能温度传感器（至少要像实际情况那样将评估板作为硬件平台）。尽管这只是一个基本设计，但是这里所描述的应用技术却可以很容易地扩展到更复杂的应用中，比如温度控制与检测应用。

和任何应用一样，第一步就是指明所需的系统功能，这项工作我们将在6.3.1节完成。掌握了系统性能要求后，下一步就是针对传感器设计相应的信号调理电路，确定数字滤波要求，并且设计所需的数据分析算法，这些内容将在6.3.2节中介绍。最后，传感器还应根据预先定义的协议将其检测结果传输至其他系统，这部分内容也将在6.3.2节介绍。

6.3.1 系统指标

我们的目标是开发出通用的温度传感器硬件和软件平台，为了避免所设计的结果只能用于少数情况，这里提出的系统指标都不太详细。尽管如此，读者应当意识到，传感器系统的指标通常是非常详细的，工程师必须在进行详细设计前仔细地审核它们，以确保这些指标能够实现。

这里开发的温度传感器可满足如下功能要求。

（1）有两个采样通道，一个是热电偶通道，另一个是冷结补偿通道。

（2）每个通道的采样速率为500次/s，这里假设温度信号的最高频率为15Hz（近似为33倍过采样）。

（3）支持J型和K型热电偶。

（4）对每种热电偶进行冷结补偿。

（5）能够指示热电偶开路和反接故障。

（6）对采样信号滤波以去除电源线噪声。

（7）允许用户通过RS-232串行端口（通信协议为38.4Kbit/s，8个数据位、1个停止位、无校验和无流控制）实现以下功能：

185

（a）标定各个通道；

（b）指定各个通道的温度报警上下限值。

（8）通过串行端口报告所有通道的温度测量值。如果系统检测出某个热电偶出现故障，那么就报告该通道故障信息，而不报告该通道的温度读数。温度测量值和故障报告采用文本格式。

（9）发现温度越界时报警。如果测量值低于温度下限值，显示传感器温度为“– – – –”，并且点亮演示板的LED1。如果测量值高于温度上限，显示传感器温度为“+ + + +”，并点亮LED2。

为了减少读者需要实现的电路数量，本应用仅检测两个通道的温度信号。从性能角度看，系统可以检测dsPIC30F6014A芯片所支持的任何数目的通道，最多可达16个。尽管采样速率的绝对值看起来有点儿高，但是这可以使我们很容易地去除电源线噪声，并且不会对信号造成明显的延时。对于所支持的通道数，如果有必要还可以提高采样速率，dsPIC系列DSC有充足的处理能力实现这一点。还要注意的是，所测温度值的报告速率非常慢，原因是报告数据可能是供人查看的。如果该数据是由系统中的其他电子设备读取，那么最好采用速度更快的二进制数据的协议，附属资源中含有相关代码。

6.3.2 传感器信号调理

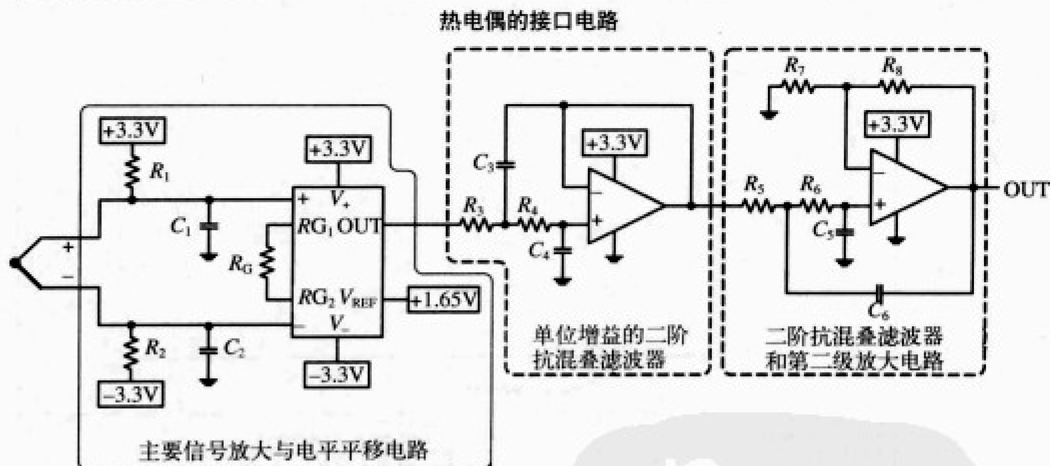
正如本章前面介绍的，热电偶传感器的接口需要完成以下任务。

（1）缓冲和放大热电偶的输出信号。

（2）实施冷结补偿。

（3）检测传感器故障。

此外，调理电路还应包括模拟抗混叠滤波器，以便充分消除250Hz以上的频率成分（由于采样速率必须至少是温度信号中最高频率分量的两倍，因此500Hz采样速率能够正确采样的信号最高频率为250Hz）。由于我们采样到的温度信号的最大带宽被假设为15Hz（这意味着实际采样速率是理论所需的采样速率的33倍），因此需要使用该信息进一步将抗混叠滤波器的截止频率压缩至15Hz。图6-8所示电路可以对单通道实现上述调理功能。下面将讨论该电路的各部分，以便更好地理解它具体包含了哪些电路，以及各部分电路的功能。请注意，该电路采用3.3V供电设计。如果需要在dsPICDEM 1.1 GPDB（它采用5V供电）上实现该电路，那么就要将所有的参考电压从3.3V变为5V，所有的－3.3V变为－5V，并将1.65V变为2.5V。 186



注意

（1）R_1和R_2提供输入共模电流通道，利用它们就能确定热电偶是否与电路相连。R_1和R_2的电阻值必须相等（取值范围是10kΩ至大约100kΩ），以防止承载输入阻抗特别小的信号，并为输入共模电流提供一个阻抗足够低的通路

（2）仪表放大器的增益由R_G决定，增益公式为

$$Gain_1 = 1 + (50\ \text{k}\Omega / R_G)$$

（3）C_1和C_2可以去除高频输入噪声。接在两个输入信号上的电容可以保持平衡的差分输入阻抗

（4）抗混叠滤波器具有巴特沃思（Butterworth）滤波器的频率特性。最后一级滤波电路的增益用于补偿仪表放大器无法达到的输出电压，通常可将信号放大到电源轨附近。其增益由下式决定

$$Gain_1 = 1 + (R_8 / R_7)$$

图6-8 热电偶接口的原理图（单通道）

1. 差分放大器

仪表放大器INA326能对热电偶差分信号进行缓冲和放大。由于仪表放大器具有极高的输入阻抗，并且本质上是差分驱动的，因此特别适合于这种应用。请注意连接在热电偶每个信号引脚上的偏置电阻具有热电偶开路检测功能，并为仪表放大器提供正常工作所需的直流

偏置通道。

2. 抗混叠滤波器

抗混叠滤波器只有一个主要任务：去除所采样的信号中高于采样频率一半的频率成分。[187] 乍一看，我们可能需要使用高阶的模拟滤波器实现该功能，以便实现极强的滤波功能，但是很多情况下，该方法并不是最好的。在前面几章中，我们暗示过模拟滤波器的缺点（有漂移、成本高和占用板空间大），并且这些问题会导致滤波电路变得更复杂。相反，可以先使用级联的二阶巴特沃思模拟滤波器实现滤波，然后通过DSP滤波器实现平移。这种滤波器[1]结构在整个通带内都有优化，并且具有相对陡峭的滚降特性（20dB/十倍频程），具体参见图6-9。

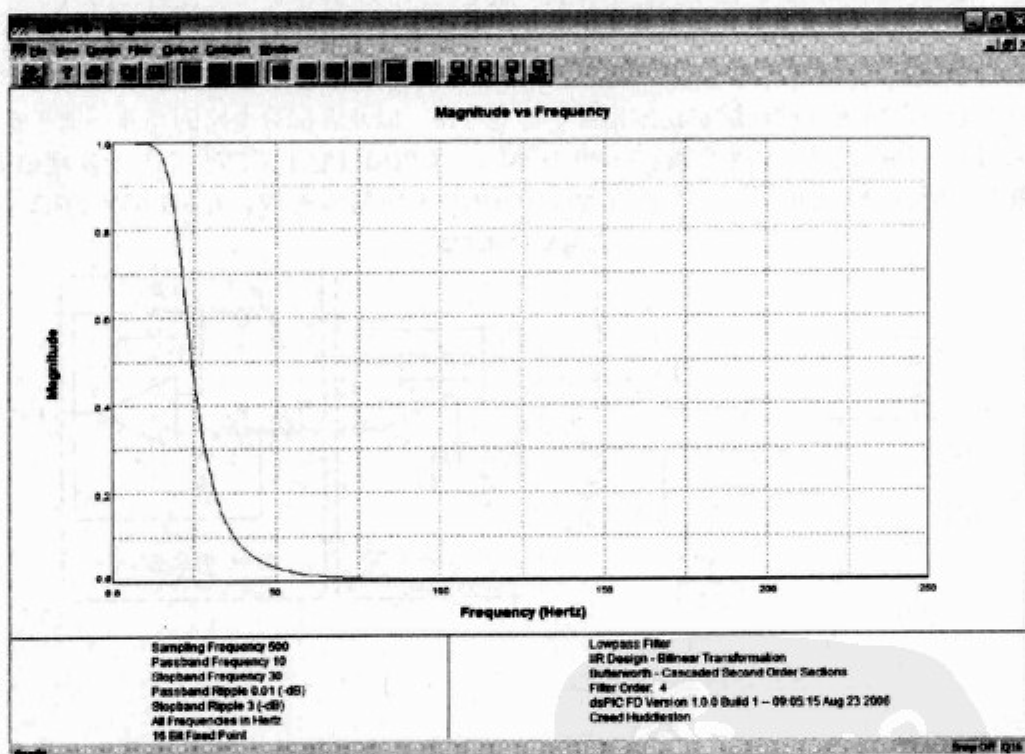


图6-9 巴特沃思滤波器的频率响应

3. 数字滤波分析

在本例中，电源信号被视为带外噪声，于是可以利用一个边沿陡峭的低通数字滤波器降低温度信号中的噪声。如果温度信号的带宽更大，那么还需要使用陷波器，这将在下一章介绍。

设计数字滤波器的软件包有很多，但是本书采用的是Microchip Digital Filter Design System v1.0.0。和本书使用的其他开发软件一样，该滤波器设计软件包也是由Microchip公司[188] 提供的[2]。利用软件包设计滤波器的优点之一是能够容易而快速地得到有关滤波器特性的图

① 见http://www.daytronic.com/reference/aliasing.htm，这有关于使用巴特沃思型抗混叠滤波器的简明扼要的讨论。

② Microchip公司的完整版Digital Filter Design软件的产品编号为SW300001。而功能简化版则便宜一些，区别是：简化版不支持较多抽头的FIR和IIR滤波器，并且不支持MATLAB。在大多数情况下，简化版软件也已经足够，其产品编号为SW300001-LT。

形化反馈。例如，图6-10和图6-11分别展示了一个宽松的和一个紧凑的阻带纹波要求。正如我们看到的，紧凑的纹波要求还会影响通带的纹波和通带窗口的宽度，它会使通带纹波更大、通带变宽并且使通带和阻带的过渡更陡峭。

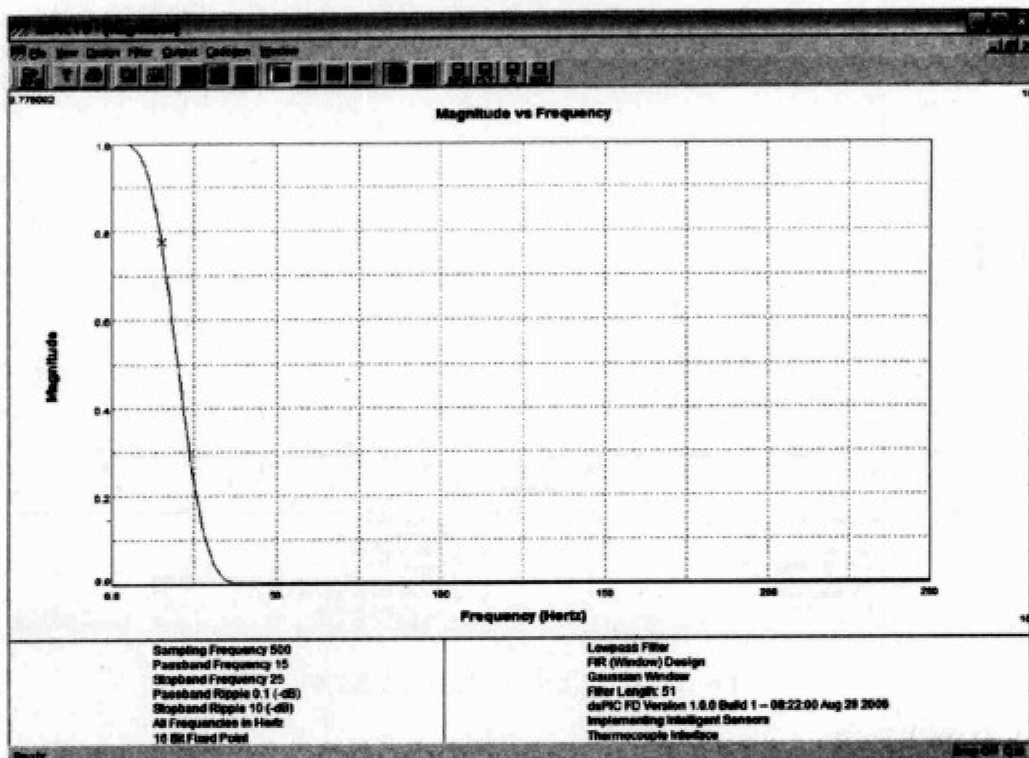


图6-10　具有宽松的阻带纹波要求的滤波器的响应

具有宽松阻带纹波要求的滤波器的通带更平坦（这可以从图6-10所示的幅频特性图中左上角看出），因此开始可能看起来可能滤波效果更好，但是该滤波器在6Hz时只允许输入信号的78%通过。相反，尽管具有紧凑阻带纹波要求的滤波器在通带和阻带都具有更大的纹波（这并不是我们一定要求的，但是仍然符合指标要求），但是它在6Hz处允许更多的（82%）信号分量通过，并且在通带和阻带之间具有更陡峭的过渡。上述分析表明，滤波器设计也是一个折中的过程，设计师必须在某一段的性能更好而另一段的性能更差之间折中。

189

在本应用中，我们将采用宽松滤波器要求，因为它能满足我们的要求，并且不会严重影响信号。设计师通常都要掌握一些滤波器理论（只要足够解决问题即可），因为这可以避免过度修改原始信号。从这一点看，它类似于医生们警告的“首先，要确保无害”[①]。

数字滤波器设计中还有一点需要注意。在上述两种情况下，滤波器的抽头数都是51。如果我们再增加抽头数，就还能提高滤波器的性能，但是这会使信号通过滤波器时出现更大的时延。在本应用中，我们有意地选用抽头相对较少的滤波器，以便系统的响应更快。

① 尽管“首先，要无害”经常被误认为是希波克拉底（Hippocrates）的名言，但是实际上，它并不是出自希波克拉底誓言（Hippocratic Oath）。但是，这句话可能出自他的著作*Epidemics*（《流行病》），他在其中写到医生必须“从两种角度来看待疾病，要看到疾病的益处或者说无害的一面”。引自http://ancienthistory.about.com/od/greekmedicine/f/ HippocraticOath.htm。

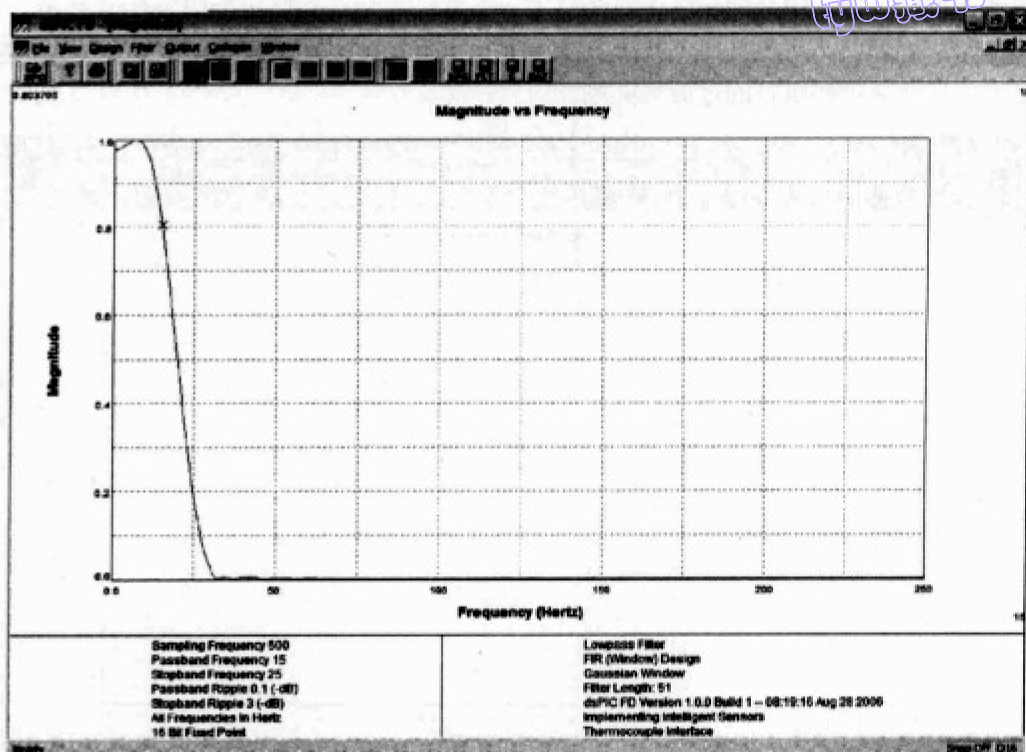


图6-11　具有紧凑纹波指标的滤波器的响应

4. 数据分析算法

本应用所需的数据分析算法非常简单，只要将滤波后的温度采样值与用户设定的温度报警上、下限值进行比较。一旦发现超出某个限制，就点亮相应的报警灯。图6-12展示了该数据分析算法的流程图。尽管这里展示的分析十分简单，但是请注意，我们很容易将它扩展成更复杂的分析算法，或者增加输出控制算法。设计人员应当始终努力开发出这种结构的算法，使之具有简单、清晰且易于扩展的特性。这不仅能使软件框架更具可读性，并且还便于为成熟的产品添加新特性。

190
191

5. 通信协议

应用设计师有时会错误地将工作在正常情况下与工作在各种情况下（包括通信接口所用物理媒介的故障）的通信接口等同起来。一名优秀的设计师会考虑所有可能出现的情况，他们开发出的系统能够稳妥地处理意外情况。这种“优雅降级”（graceful degradation）特性极为重要。即使系统无法纠正错误，它也应该向主机告警，然后采取任何必要行动以确保系统处于安全的状态。此外，还应实现一种基于状态的通信接口，一旦遇到可能造成系统出现故障的意外输入，它能在任何时候自动复位。

第一个应用将采用简单的、用户可读的基于标准RS-232串行端口的通信协议。该方法不但允许用户与系统进行实实在在的交互，而且还允许用户逐步深入（不是贸然介入）基于状态的通信处理器的内部。其中后一点更为重要，因为极大量的产品都是由很脆弱的、不可靠的通信接口来实现的。在用户看来，没有什么比无法解释的系统故障（通常表现为系统“死机”）更让人懊恼的事情了。这种由于所实现的通信接口太脆弱而导致的故障，都是不可原

谅的。

既然已经吹捧了基于状态的通信处理的优点，下面就仔细看看它到底是什么意思以及如何利用该方法开展设计。在基于状态的处理器中，应用系统执行的动作取决于处理器当前的状态（其实是一个存储在全局变量中的值）以及当前处理的数据值。根据所采取的动作，状态值可能改变也可能保持不变。下面通过举例来说明该概念。

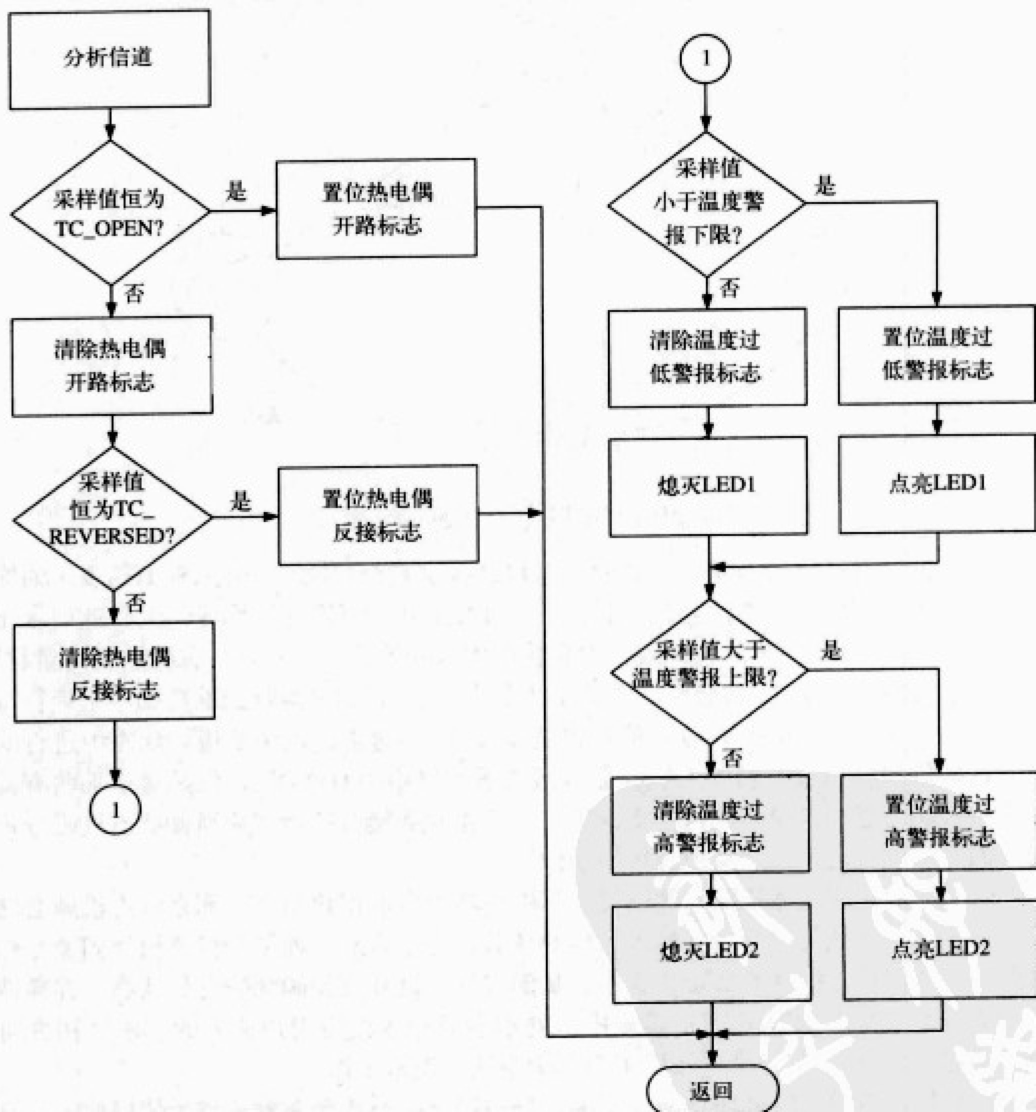


图6-12 热电偶传感器的数据分析流程图

图6-13显示了一个简单的通信处理器的状态图。图中每个圆圈代表一种处理状态，而从每个处理状态延伸出来的曲线被称为*转换条件*，它是系统从一个状态转换到另一个状态所需要满足的条件。如果某个转换线上未给出条件，那么状态机就会在执行完初始态的所有操作后自动移向终态。例如，请看图中的*初始化状态*，有一条转换线延伸至*接受输入状态*，而该转换线没有相关的条件，这就意味着状态机在完成*初始化状态*的所有操作后就会立即转向*接受输入状态*。

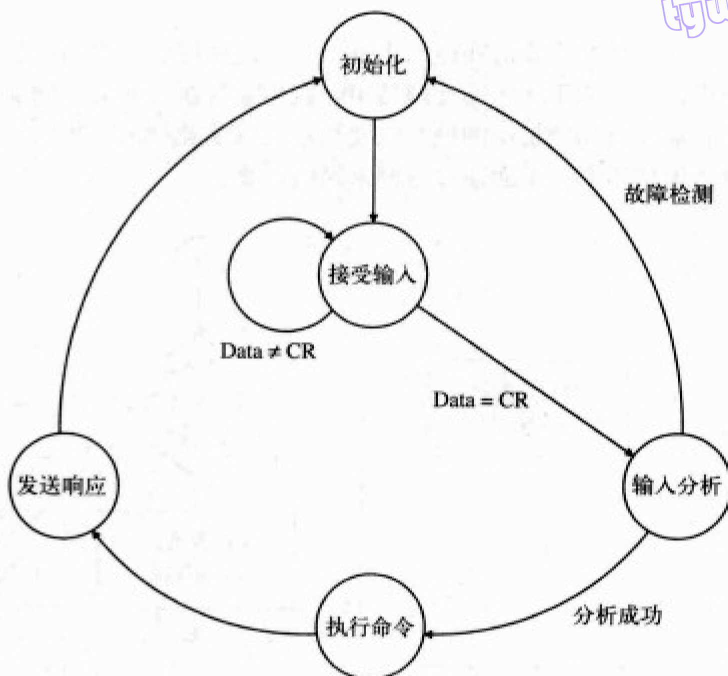


图6-13　简单的通信处理器的状态机

一旦状态机进入接受输入状态，它就会保持该状态直至接收到一组ASCII码表示的换行符输入信号。图中有两条转换线指出了这点，一条是Data ≠ CR的转换线，它将返回接受输入状态，另一条是Data = CR的转换线，它将转向输入分析状态。一旦状态机接收到新数据，就会在接受输入态进行处理，然后要么保持在接受输入状态（如果接收到的数据不是换行符），要么转向输入分析状态（如果接收到的数据是换行符）。通常，在接受输入状态中进行的处理会沿着这些转换线将新数据加入缓存器（假设缓存器中还有空间），但是对于那些有特殊要求的应用可能会更加复杂。在任何情况下，一旦接收到换行符后就应通知状态机去分析接收到的数据，以确认用户希望系统进行的动作。

如果输入分析处理能够从输入数据缓存器中成功地提取用户命令，那么状态机就会转换到执行命令状态，并执行用户命令输入指定的任务。另一方面，如果分析器检测到命令输入中有错误，就会使状态机返回初始化状态，从而将状态机复位到确知的良好状态，并等待用户的下一条命令。通过这样处理，系统操作就不会受到错误的用户输入的影响。用户可以（而且很可能会）出错，但是那些错误不会影响系统的正常工作。

193

执行命令状态内的处理过程只是简单地执行与指定命令及命令参数相关的操作。一旦完成任务，状态机就会转向下一状态——转换响应状态，汇报命令执行的结果。最后，给主机发送完命令响应后，状态机又返回初始化状态，重新初始化所有的状态机变量，准备接收下一条命令。

尽管状态机也可以用硬件实现，例如用可编程逻辑芯片实现，但是用软件实现会更加方便、灵活，而且成本更低（当然，要假设处理器能够以足够快的速度循环处理各个状态，以满足所有的时序要求）。尽管软件状态机的设计很简单，但是有些指导可能对那些从没有设计过软件状态机的读者有帮助。

在实现软件状态机时，应用系统应使用全局状态变量来跟踪机器处于哪个工作状态，并记录当状态机跳转到另一个状态时需要保存的信息。状态机本身至要位于最高级，通常在一个函数中实现它，并且在新输入数据到来时被反复调用，按照期望的顺序循环执行。如果应用程序采用C语言设计，那么该函数通常采用switch语句，而声明的参数就是当前的处理状态。以我们已经开发的例子为例，相应的代码如下所示。

代码实例6-1 简单的通信状态机实现

```
                    //  Communication State Definitions
#define COMM_ST_INIT            0         // Initialize state machine
#define COMM_ST_ACCEPT_INPUT    1         // Accept input data
#define COMM_ST_PARSE_INPUT     2         // Parse complete line of input
data
#define COMM_ST_EXEC_CMD        3         // Execute the parsed command
#define COMM_ST_TX_RESPONSE     4         // Transmit the command response

                         //  State Variables
Uint8
    g_ui8CommState = COMM_ST_INIT;  // Global communication processing state

/**************************************************************************
*   FUNCTION:      CommProcRxData(Uint8 ui8NewData)                      *
*                                                                        *
*   DESCRIPTION:   This function implements the high-level software     *
*                   state machine for communication processing. It is   *
*                   called by the main processing loop whenever the     *
*                   processor receives new input data, and the function *
*                   cycles through the appropriate states based on the  *
*                   current processing state and the new input data.    *
*                                                                        *
*   PARAMETERS:    ui8NewData - new input data to process               *
*                                                                        *
*   RETURNS:       The function returns the new communication processing *
*                   state.                                               *
*                                                                        *
*   REVISION: 0    v1.0              DATE: 12 June 2006                  *
*       Original release.                                                *
**************************************************************************/

Uint8
    CommProcRxData(Uint8 ui8NewData)
    {
    //  Local Variables
    Bool
        bProcessingState = TRUE;    // Continue processing state flag (set to
                                    //   continue processing states, cleared
                                    //   to stop processing after the current
```

194

```
                                        //   state)
    //  Identify the current communication state
    //  and process the new data accordingly
    while (bProcessingState)
        {
        switch (g_ui8CommState)
            {
            case COMM_ST_INIT:
                //  Initialize the comm state machine
                ...                     // Perform state-specific processing
                bProcessingState = FALSE;   // Finished processing for this cycle
                //  Cycle to next state
                g_ui8CommState = COMM_ST_ACCEPT_INPUT;
    break;
case COMM_ST_ACCEPT_INPUT:
    //  Accepting new input data
    ...                     // Perform state-specific processing
    //  Cycle to next state
    if (ui8NewData == '\r')
        g_ui8CommState = COMM_ST_PARSE_INPUT; //  Parse the
                                              //    data if
                                              //    we've
                                              //    received
                                              //    the entire
                                                    line
    else
        bProcessingState = FALSE;     // Finished processing for this
                                      //    cycle
    break;
case COMM_ST_PARSE_INPUT:
    //  Parse the complete input data buffer
    ...            // Perform state-specific processing that
                   //   returns a value of bError to indicate
                   //   whether the parsing was successful

    //  Cycle to next state based on
    //  the success of the parsing
    if (bError)
        g_ui8CommState = COMM_ST_INIT; // Error detected so
                                       //   reset
                                       //   the state machine and
                                       //   wait for next command

    else
        g_ui8CommState = COMM_ST_EXEC_CMD;      // No error so
                                                //   execute the
```

```
                                                    //   command
    break;
case COMM_ST_EXEC_CMD:
    //  Execute the parsed command
    …                      // Perform state-specific processing
    //  Cycle to next state
    g_ui8CommState = COMM_ST_TX_RESPONSE;
    break;
case COMM_ST_TX_RESPONSE:
    //  Transmit the command response
            …                      // Perform state-specific processing
            //  Cycle to next state
            g_ui8CommState = COMM_ST_INIT;
            break;
        default:
            //  Should NEVER get here so reset
            //  the state machine
            g_ui8CommState = COMM_ST_INIT;       // Cycle to next state
            break;
        }
    }
return g_ui8CommState;
}
```

该软件状态机代码中有很多地方需要注意。比如，我们使用全局变量（g_ui8CommState）保持处理机的处理状态。代码会将g_ui8CommState初始化为变量声名中的状态，以确保状态机从确知的状态（初始化态）开始工作，并且该函数会在状态机需要转向新状态时更新g_ui8CommState的值。当程序执行完当前状态的所有操作后就会返回状态机的终态，但是由于当前状态的值已经存放在全局变量里，因此返回值只是起到发送消息的目的。为了确保工作正常，只有CommProcRxData()函数可以改变g_ui8CommState变量。尽管其他子程序也会通过检查CommProcRxData()函数的返回值来掌握系统状态，但是它们都无法改变该变量的值，因为如果允许它们修改就会导致深层次而且隐蔽的软件交互作用，这可能会导致将来发布的新版本状态机出现问题。

图6-13所示的状态图显示的部分状态，需要输入或者在特殊条件下才会出现的状态才能转入下一个状态。根据这里所示的结构，状态间的功能循环并不需要任何用户输入，而只需调用ComProcRxData()即可。根据必须完成的处理工作量以及处理所需的时间，可能还需要对该方法做适当修改。

6.4 硬件实现

图6-14所示的硬件框图给出了本节将要讨论的主要硬件的组成框图。请注意，这里只给出了一种电路设计，但是这决不是唯一的，那些具备更多模拟设计技巧的读者可以在这里给出的电路的基础上进行改进。

195～197

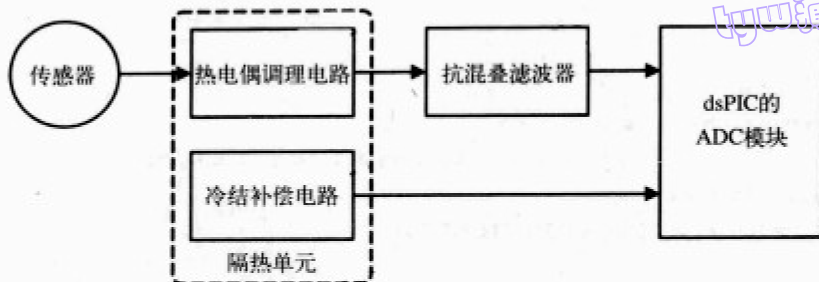


图6-14 硬件组成框图

6.4.1 模拟放大器和抗混叠滤波器

图6-8是模拟放大器和抗混叠滤波器部分的原理图。这里的放大器由单片仪表放大器实现，其输出被送入一个四阶的巴特沃思低通抗混叠滤波器，该滤波器由2个轨到轨运放以及一些无源器件构成。设计该电路时有三个要点需要考虑。

（1）使电路的功耗最小，以减小自发热的影响。

（2）清洁、简化印制电路板的布局，使串扰和噪声最小化。

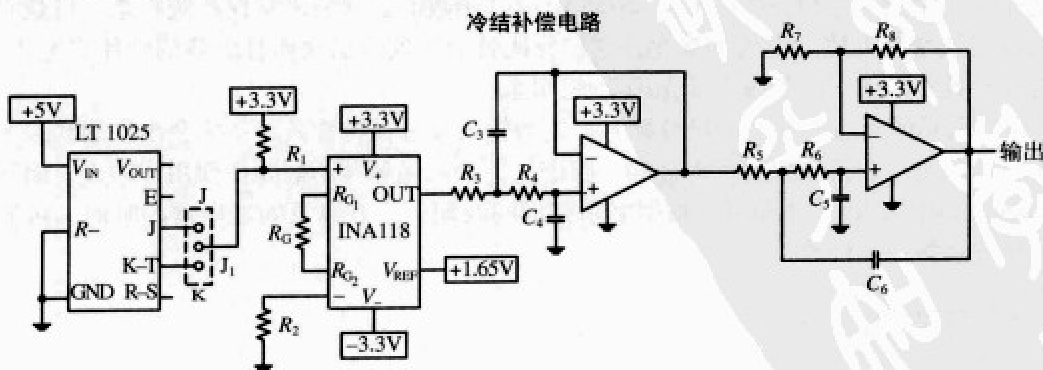
（3）采用良好的电源旁路设计，以减小噪声。

主要通过采用低功耗的IC（很多供应商都有相应的产品）实现低功耗设计。由于热电偶的输出是弱驱动、低电平信号，因此电路布局很关键。连往热电偶引脚的走线也要尽可能短，并且如果可能，通往同一个热电偶的导线都要并排一起走线，以便提高共模抑制（这能保证热噪声和电气噪声等量地耦合进两个引脚中）。最后，采用具有低纹波和低漂移的高性能固态电源也很重要，否则电源本身的噪声（它与热电偶输出中的噪声不同）也会被引入到系统中。

6.4.2 冷结补偿

尽管实现冷结补偿的方法有很多种，但是这里的电路采用了冷结补偿芯片（Linear公司的LT1025）实现硬件功能，图6-15是基于该芯片的冷结补偿电路的原理图。通过改变J1的跳线位置，可以选择是J型还是K型热电偶补偿。

198



注意

（1）热电偶信号处理通道所用电阻和电容值的大小要尽可能匹配

（2）跳线J_1与仪表放大器的J脚相连时，选择J型热电偶，当与仪表放大器的K脚相连时，选择K型热电偶

图6-15 冷结补偿电路的原理图

这里，LT1025的电源输出经缓冲后被dsPIC系列DSC测量，然后再通过软件算法去除所测热电偶测量值中的相应的冷结电压。为了达到最高的精度，可以在每个热电偶输入通道使用一个LT1025。但是，实际中只要将LT1025放在两个热电偶的引线对之间（即将一个热电偶的一对引线放在LT1025的一侧，而将另一个热电偶的一对引线放在另一侧），那么用一片LT1025就能处理两路热电偶输入信号。当然，还可以使更多个热电偶共享同一片LT1025，但是冷结补偿芯片距离实际的热电偶引脚越远，补偿的有效性就越差。

6.4.3 信号隔离

信号隔离的等级要求取决于传感器工作的环境。在很多情况下，由于传感器并不是工作在恶劣的或者特别敏感的环境中，因此隔离要求相对较低。但是在其他情况下，例如当热电偶被放置于电驱动发热元件附近或者被用于测量人体温度时，应用系统就需要更高的隔离等级，以保护传感电路或者被测系统。

199

正如我们料想的，隔离热电偶信号的方法有很多种，它们具有不同的成本、复杂度以及信号衰减度。最廉价的方法只需使传感电路和被测物之间保持极高阻抗即可。比该方法成本稍高的方法是采用带有高电阻输入的运放缓冲器，它可以提供数兆欧至数百兆欧的阻抗。进一步考虑价格和性能，这种TI公司（前身是Burr－Brown公司）出品的仪表放大器INA118具有$10^{10}\Omega$的输入阻抗，并能克服输入端上的40V共模电压，这在传感器必须工作于可能有大共模电压的环境时特别有用，因为这时共模电压很容易耦合进热电偶的信号中[①]。正如我们在前面讨论过的，如果热电偶十分靠近电源线（或者其他非零电压的导线），就很容易发生上述情况。如果输入电路无法承受额外的电压，那么传感器就会被损坏。如果在芯片上集成过压保护，那么设计师就能够减小电路板尺寸、降低安装成本以及器件成本。但是，请注意，40V的共模电压保护还不足以抵抗大多数交流电源线信号，根据不同的工业标准，它们的交流电压可达110V～480V。如果系统必须抵抗这么高的电压，那么就需要额外的电路将最大输入电压抑制到芯片能够承受的等级。

最后，价格最贵、性能最好的方法是对热电偶信号与系统的其他部分进行光隔离。这可以通过光隔离模拟放大器（价格极贵）或者利用外部ADC对非隔离、放大后的热电偶信号数字化，然后利用光隔离的数字化接口将数据传输至处理器（价格较低）来实现。尽管仅仅对ADC接口实施光隔离要便宜很多，但是在传感器必须将电流或电压限制到监控系统期望的范围的情况（比如在医疗应用系统中）下是不允许的。

在这类应用中，我们前面采用了仪表放大器隔离法。因为它的构造简单、价格相对便宜，并具有极佳的性能。图6-8所示单通道信号隔离原理图中就使用了仪表放大器INA326。

6.5 固件实现

200

为了简化固件开发，应用系统大量使用了Microchip 16-bit Language Tool libraries来访问dsPIC系列DSC的片上外围设备，并利用滤波等DSP功能。本节将详细介绍固件各部分，以便读者更好地理解如何使用这些程序库以及使用后会对编码结构产生什么影响。

① 这里可以使用各家芯片制造商生产的很多型号的仪表放大器。TI的这款芯片简单，并且作者最为熟悉。

6.5.1 信号采样

在本应用中，我们采用dsPIC系列DSC的内部12位ADC采样热电偶数据。回忆第3章的内容就会发现，为了配置ADC模块，我们需要掌握以下信息。

（1）用作模拟输入的I/O端口管脚。

（2）采样速率。

（3）是否使用交错采样，是否需要在给定的采样周期内对所有信号转换完毕后产生一个中断信号。

（4）是否需要复用结果缓冲器。

（5）转换数据的格式。

由于使用了dsPICDEM电路板，因此有些硬件已经决定了。dsPICDEM板采用5V板上稳压器作为V_{DD}和AV_{DD}，芯片采用外部7.3728MHz晶振时钟，内部4×PLL倍频（因此实际时钟为29.4912MHz）。此外，dsPICDEM板上的部分模拟输入端口已被使用，如表6-2所示。

表6-2 模拟输入信号的指定

信号	是否已用	描述
AN0/RB0	是	ICD2的可编程数据信号（PGD）
AN1/RB1	是	ICD2的可编程时钟信号（PGC）
AN2/RB2	否	可供用户使用
AN3/RB3	是	数字电位计，已经过低通滤波
AN4/RB4	是	模拟电位计RP2—0～AV_{DD}
AN5/RB5	是	模拟电位计RP3—0～AV_{DD}
AN6/RB6	是	模拟电位计RP1—0～AV_{DD}
AN7/RB7	否	可供用户使用
AN8/RB8	是	温度传感器信号，已经过低通滤波
AN9/RB9	否	可供用户使用
AN10/RB10	否	可供用户使用
AN11/RB11	否	可供用户使用
AN12/RB12	否	可供用户使用
AN13/RB13	否	可供用户使用
AN14/RB14	否	可供用户使用
AN15/RB15	否	可供用户使用

请注意，仅当跳线器J8有效时，AN0和AN1才被分别用于PGD和PGC。如果这些跳线器被去掉，那么应用系统就可以使用AN0/RB0和AN1/RB1作为模拟输入或者数字I/O引脚。

本应用使用AN11作为冷结补偿输入，使用AN7作为热电偶输入。这些输入脚都是相对于地的单极性信号（仪表放大器已经考虑到将差分信号转换为单极性信号），并且信号被放大到模拟电源电压的满幅范围内（即AV_{SS}至AV_{DD}）。

6.5.2 数字滤波器实现

感谢Microchip公司提供了DSP 库以及dsPIC Filter Design™软件，因此在本应用中使用的简单数字滤波器设计就变成了通过Filter Design软件生成正确的滤波器系数，并利用DSP库提供的函数实现该FIR数字滤波器。尽管如此，我们需要先了解用于处理数据的函数库的

基本结构，并理解在应用中使用这些函数库的限制。这些限制包括应用函数库的一般要求以及某个滤波函数的特殊要求。

DSP函数库采用小数向量存放操作数。尽管这听起来有些学究，但是小数向量表只是一种双字节数据数组，它采用1.15数据格式表示每个数据。该函数库对小数向量有两个要求：数组成员必须是连续的（即不允许使用链表形式），并且由于它们是16位数据，因此该数组必须从偶数存储器地址开始存放。这些细节要求是由基本的dsPIC硬件决定的，并且能保证函数库使用处理器指定的DSP引擎优化数学运算。还需要注意的是，通常，这些库函数处理位于默认RAM存储器空间（*X*数据存储器和*Y*数据存储器）内的小数向量[①]。滤波函数还进一步指定哪些数据可以放入存储器。输入和输出数据可以保存在默认的RAM存储器空间（*X*数据和*Y*数据），但是滤波器系数只能存放在*X*数据存储器或者程序存储器中，而滤波器延时值必须保存在*Y*数据存储器中。[②]

202

当然，所有普通数学运算规则都可以用于处理这些向量，普通传感器编程规则也一样适用，比如开辟足够大的目的向量缓冲区，以处理指定数学运算的结果。读者必须注意，为了保证运算速度，DSP函数库不检查输入和输出数据的有效性，并且尽管函数库已经设定了饱和或者溢出标志，但其中的函数不会用它们。除非应用程序的数据特别防止发生这种情况，否则程序员必须按照如下要求在每次调用库函数时检测上述标志。

对于本应用，我们将使用以下参数：

采样频率：500Hz

通带频率：15Hz

阻带频率：25Hz

通带纹波：0.1dB

阻带纹波：3dB

滤波器类型：高斯型

在dsPIC Filter Design软件中输入上述参数，产生的结果滤波器的代码（存放在`SensorFilter.s`文件中）就会为我们提供所需的数据结构。

6.5.3 数据分析实现

本应用数据分析部分的实现非常简单，将滤波后的数据转换为对应的温度值，然后将它与温度警报的上、下限值做比较即可。尽管这种处理很基本，但是这些操作不但适用于温度监测系统，也适用于任何温度控制系统。千万别低估了简单的力量……

6.5.4 错误处理实现

和数据处理的实现一样，本应用的错误处理程序也非常简单。请注意，在这里所说的错误处理，是指那些意外的对系统有不利影响的情况，而不是温度超出数据分析软件的检测范围。它们之间的差别很小，但是很重要。当出现温度越界的情况时，它对系统来说是其实是正常的，并且不会影响系统正常运行。但是这里所说的错误条件与其有本质不同，因为，如

203

---

① 这个注释出现在*16-Bit Language Tools Libraries user's Guide*的User Consideration一节。

② 有关DSP滤波子程序如何获得存储器空间的内容，请参阅*16-Bit Language Tools Libraries user's guide*的Fractional Filter Operations一节。

果无法检测到这些错误，那么系统就会报告出无效的信息，并且这些无效信息可能会导致灾难性结果。

我们将检测在温度测量系统中常见的两种故障：一种是热电偶开路（断开），另一种是热电偶反接。这两种情况的表现有所不同，并且都会对系统造成不同程度的影响。例如，开路的热电偶基本上属于灾难性的故障，系统可以检测它，但是无法纠正。相反，系统可以补偿热电偶反接的情况。尽管（或许是正因为）热电偶开路属于灾难性故障，它相对容易被检测到。只需将原始输入信号电压与一个阈值做比较就能检测出热电偶开路故障。由于在热电偶输入端添加上拉和下拉电阻，因此开路的热电偶会使输入信号的电压变成接近满幅，这明显超出了正常的热电偶信号输出范围，从而可以判断出发生故障。

检测热电偶反接更具挑战性，因为当管脚反接时，在一定的温度范围内仍然会产生“有效的”热电偶输出信号，即该温度对应的输出电压仍位于正常电压输入范围之内。确定热电偶是否反接的唯一方法是当温度变化时看热电偶输出电压是否随之变化。正常的热电偶信号是线性变化的，其输出电压随温度升高而增大（变得更大）。如果热电偶引脚反接，那么输出电压就会随温度升高而下降（变得更小）。如果有一个可用的热源，比如，当加热元件与热电偶结对，那么系统可以打开加热器一段时间，并检查对应的热电偶的输出电压。如果系统加热后电压增大，那么接线就是正确的（至少从极性看是正确的）。如果用户有一个冷却源（比如，在冰箱应用中），也可以进行类似的测试，只要注意到启动制冷元件后，期望的热电偶输出电压会随之减小。

当没有加热源或者制冷源时，测试就会更困难，因为此时唯一的方法就是等着热电偶的输出电压在负方向（假设我们对系统加热）或者正方向（假设在制冷系统中）超出了期望的范围。尽管如此，一旦发生上述情况，固件就能标记错误并向用户报告。

204

6.5.5 通信协议实现

本书三个应用实例所采用的通信协议已经在第4章讨论过。这里我们利用UART1通过RS-232接口将数据传输至主机，并且由于协议的基本命令结构支持各种应用，因此这里不必对其进行扩展。既可以通过用户可读的文本协议也可以通过机器可读的二进制协议实现通信，为解释这个过程，实例代码允许设计人员使用任何一种技术，只需简单地注释掉`ProtocolDef.h`头文件中的相关内容即可。

处理数据的基本方法是在主处理循环中通过`CommIsRxPending()`函数检查接收到的数据，然后，如果有新的串行数据，就利用`CommGetRxData()`函数读取，并利用`ProtocolProcRxData()`函数处理新数据。该函数会分析接收到的数据，并且当接收到完整的命令消息时，它会通过`ProtocolProcMsg()`调用适当的命令处理器。在识别命令后，`ProtocolProcMsg()`又会调用相关的底层命令处理子程序，并利用命令指定的响应函数向主机发回适当的响应。一旦这些响应消息发送完毕，`ProtocolProRxData()`函数又会复位分析状态机，并等待下一条消息。

6.6 小结

本章已经进入传感器设计领域。现在我们能够数字化模拟传感器信号，对其滤波以去除不想要的噪声，并分析滤波后的数据并提取有用的信息，还能将这些信息传送至系统的其他

部件。本章以前面设计的框架作为基础，开发出更为复杂的系统。后面还会继续使用该框架作为设计基础。下一章我们还要继续扩展前述框架，并完成第二个应用设计——压力/称重传感器系统。

第7章 传感器应用——压力和称重传感器

从前一章的温度检测应用设计中，我们积累了一些传感器系统设计方面的经验。下面将在此基础上扩展基本的传感器框架以实现称重传感器和压力传感器。称重传感器可以测量物体的重量，压力传感器能测量一定范围内的压力；这两者已被广泛应用于各种领域。它们能根据重量来计算微型电子器件的数量，能测量病人的血压，还能控制施加到金刚钻上的压力。无论你是否意识到，称重和压力传感器已经融入到我们的日常生活中。

7.1 称重和压力传感器的类型

在深入讨论该题目前，必须认识到称重传感器和压力传感器在本质上是一样的。它们都是测量传感器上的载荷（或重量）。唯一的差别是压力传感器的输出与传感器单位面积上的重量成比例。例如，一种常用的压力传感器采用了圆柱型封装，它能保证在高压环境中使用时不损坏敏感元件。当压力传感器工作时，其输出信号反应了施加在圆柱体平面上的力（假设传感器安装正确）。如果已知传感器的直径，那么就可以根据该装置的测量值计算出对应的压力。

举例可以帮助读者理解压力的计算过程。假设有一个上述的圆柱体传感器，该圆柱体的直径为0.125英寸[①]（1/8英寸），如果传感器测得的压力为100磅[②]，那么对应的压力就可由下式计算出来：

压力 = 载重/面积

= 载重/ (πr^2)

= 100磅 / [(3.14)(0.125英寸)^2]

= 2038（磅/英寸^2）

如果，同样的100磅压力施加在直径为上述圆柱体传感器直径一半的传感器上（即直径为0.0625或1/16英寸），那么对应的压力为：

压力 = 100磅 /[(3.14)(0.0625英寸)^2]

= 8153（磅/英寸^2）

如今在测量称重和压力检测控制应用中，最常用的传感器有应变计和压电传感器两类。尽管下文没有详尽讨论这两种传感器，但是它能为我们提供与其接口有关的信息，并体验一下这些传感器的作用。

7.1.1 应变计

简而言之，应变计是一种自身电阻随施加于其上的压力改变的器件。该效应最早是Lord Kelvin于1856年发现的，但是由于应变计的电阻随着压力变化很小，即使在负载发生巨大变化时也是如此，因此直到1938年它才真正被应用。有趣的是，在1843年英国物理学家

① 1英寸=2.54cm。——编者注

② 1磅约等于254g。——编者注

Charles Wheatstone爵士设计的一个完美的电路能准确测量这种微弱的变化（见图7-1）。

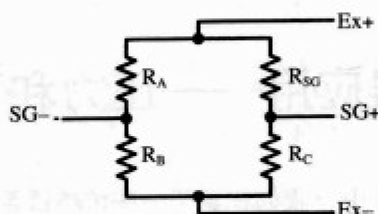


图7-1　惠斯登电桥应变计的原理图

该电路中，Ex+和Ex−与电路的外部激励电压源相接，SG+和SG−是需要测量的差分电压输出。这种电阻布局所产生的输出电压与电桥上每个桥臂的电阻值的变化成比例。如果我们令SG−桥臂上的电阻值（R_A和R_B）与SG+桥臂上的电阻值（R_{SG}和R_C）对应相等①，那么在SG+和SG−两端测得的输出电压等于零。这看起来毫不奇怪，但是由于电阻的微弱变化都会导致电桥输出电压不等于零，因此我们能够很容易地跟踪电桥中某个元件的电阻变化。可能有些读者已经预见到，如果将电桥中的某个电阻换为应变计元件，那么就能精确地测量由于施加到应变计上的力所引起的电阻变化。

210

*称重传感器*就是通过在电桥上放置应变计，从而测量施加到某个机械轴上的外力的。上一章所介绍的热电偶是所谓的*无源敏感元件*，因为被测参数的变化（在那里就是温度）会产生传感器输出信号能量。相反，称重传感器则是*有源敏感元件*，被测参数的变化会调节外部*激励*电压源。用户可以通过选择外部激励电压源的特性优化参数测量。这对用户来说增加了设计的灵活度，但是它还需要额外的电路来提供激励电压，并增加了系统发热问题，从而又会降低传感器读数的精度。

和热电偶一样，应变计/称重传感器的输出电压非常微弱，通常每伏激励电压只能产生数毫伏输出。这就要求仔细进行信号调理，但是我们可以使用电桥输出的比例特性来增强测量小信号的能力。由于应变计易于使用并且相对便宜，因此应变计型称重传感器最常用，这也是我们将在本章的应用设计中采用的类型。

7.1.2　压电传感器

第二种流行的称重传感器是压电传感器，它是基于某些材料在受力时能够产生少量电荷的特性而制成的②。这种输出信号非常微弱，通常只有数纳库仑③，因此需要使用特殊的电路——*电荷放大器*——将输出电荷的微小变化转变为外部系统能够检测的电压信号。

由于压电传感器对称重快速变化的响应比应变计好，因此压电传感器通常应用于称重物体运动的场合，即在正常范围内变化。但是，如果测量系统的响应速度比传感器本身慢很多，该优点并不一定能得到更好的读数。此外，当施加在传感器表面的是恒力时，传感器输出的电荷会随时间而逐渐消逝，因此称重相对来说是一个常数，这种情况下压电传感器通常难以测准。

211

① 即$R_A=R_{SG}$，$R_B=R_C$。——译者注

② 该属性被称为压电效应。

③ 1纳库仑=10^{-9}库仑。

但是，电荷放大器价格昂贵并且工艺复杂，这都限制了压电传感器的应用。在那些需要极高频率响应的场合，为达到应用系统的性能要求就必须为使用压电传感器而付出这些额外代价。

7.2 称重测量的要点

一般来说，称重测量的要点同其他传感器系统类似，包括测量的范围、分辨率以及精度。尽管每个应用中称重测量面临的挑战和信号特性都不同，但是像线性化以及标定等方面的注意事项和其他传感器基本相同。

7.2.1 测量范围

称重测量应用需要覆盖很宽的称重值范围（从数毫克到数千千克），因此设计人员需要确定所设计的系统能承受的合理的称重范围。很多工程师都会忽视的一点是（至少在第一次会忽视），必须指明正常工作范围和最坏情况下的动态范围指标。一个包装称重系统为测量包装从数英尺高坠落时承受的冲击力，就要求测量范围达0～200磅。这不仅是出于机械考虑（要求系统能够承受撞击而不会损坏），而且还要求信号调理电路产生的瞬间电压信号不会损坏电子器件。

例如，我们可能设计了一个正常工作范围是0～20 000磅/英寸^2、最大测量值是30 000磅/英寸^{2}的压力传感系统。该系统能够测量注模机的真空腔内的压力（即模具内部的压力）。选择工作范围为0～20 000磅/英寸^2，基本上就意味着要在该范围内保持最高的精度，因此就可以选择这两个端点作为标定点。

7.2.2 测量分辨率

假设要求输入信号放大后的最大值和最小值分别为AV_{SS}和AV_{DD}，那么测量的分辨率就等于该最大范围除以所使用的ADC的量化等级（10位ADC有1024个量化等级，12位ADC有4096个，而16位ADC则有65 536个）。这里，我们将使用dsPIC30F6014A的12位片上ADC，因此共有4096个量化等级。对于30 000磅/英寸^{2}的测量范围（请记住，应使用最大测量范围而不是正常测量范围），对应的测量分辨率就等于7.3（磅/英寸^2）/等级[30 000（磅/英寸^2)/4096=7.32（磅/英寸^2）/等级]。

212

如果需要更高的分辨率，那么就得采用分辨率更高的片外ADC。然而，对于大多数注模应用来说，12位的分辨率已经足够了。

7.2.3 测量精度

另一个需要考虑的是测量精度。这取决于从传感器到ADC的整个信号链路，必须仔细考虑模拟信号调理电路、电压激励源、ADC的精度以及由于温度变化引起的信号衰减等。信号链路中的不精确性是会积累的，因此其中的每个环节都很重要。比如在最开始，桥式称重单元的精度可达到0.1%左右，但是这还与所选的器件有关。

7.2.4 挑战

桥式称重传感器与热电偶信号面临的挑战基本类似，主要是输出电压信号的幅度极小

（至少在载重较小时很小），并且信号精度受温度影响而下降。

1. 信号特性

前面已经涉及了一些有关称重传感器的信号特性的内容，就是所谓的输出电压低（通常只有数毫伏到数百毫伏）。这个信号范围还与外部激励电压源有关，但是设计师通常不得不在使用高电压激励源所带来的优点和由此引起的自发热问题的缺点之间进行折中考虑。

尽管称重传感器的频率响应是传感器的固有特性，但是它们通常可用于测量频率为几十赫兹到数百赫兹的信号。如果采用保守的5倍采样速率来采样该信号，那么根据不同的应用，设计师就能得到频率就为100Hz～1kHz或者更高的采样信号。显然，只要称重值在采样周期内不发生变化，就可以在很低的采样速率下进行采样。

213

2. 热补偿

自发热问题是采用大电压激励源以及散热通道受到限制时（比如在密封的外壳里）最需要强调的问题。由于我们的应用采用5V的低激励电压源，因此系统的自发热问题不必担心。然而，在那些必须考虑自发热的应用中，可以通过采用小占空比的脉冲式激励来降低自发热的影响。在这种情况下，信号采样必须与激励电压的占空比同步，以确保仅在激励电压源“导通”时采样。这种技术还可以用于大电压激励源场合（比如10V），但是必须保证该激励电压不会损坏其他任何电路。比如，有些应用采用激励电压作为参考电压，那么对于dsPIC的ADC来说，该激励电压就受到dsPIC系列DSC的电气特性中参考电压一项的限制。即使在采用更高的激励电压时，传感器的输出信号本身符合ADC的有效输入范围要求，也可能因为激励电压不符合ADC参考电压的要求而不能使用。

3. 线性化

与热电偶传感器不同的是，由于电桥的输出在整个称重传感器工作范围内都具有极好的线性度，因此基于应变计的称重单元的线性化并不困难。这极大地简化了设计任务，设计师可以使用各种线性化硬件或者软件。

4. 标定

根据不同的应用，基于应变计的称重传感器的标定可能很难实施。在某些情况下，特别是在称重时，设计师可以假设来自被称物体的重力是垂直于地面的。假设最大称重也是“允许的”（这是公认的不严格定义），那么用户就能利用测量范围内最大值和最小值附近的两个标准重物来标定传感器系统。标定系统会记录传感器在每个标定点处的测量值（即传感器的输出电压），并计算直线标定增益和偏置量，这和我们在标定热电偶系统时所采用的方法完全相同。

遗憾的是，我们通常无法如此轻易地就能标定称重系统，一方面是因为称重不合理（比如，在大多数情况下都无法将称重变化为100磅或50 000磅作为标定点），另一方面是因为被测系统的物理特性不允许使用已标定的称重。比如，注模机系统就属于后一种情况，它通常含有很多垂直安装的传感器来测量模具中塑料融化物的压力。这就不可能（或者说最不可行）在这种传感器上施加一个已标定的20 000磅/英寸^{2}的称重物。因此，我们必须寻找替代的方法。

214

在这些情况下，有一种十分普通的技术，它通过在称重传感器输出端连接一个阻值已知的电阻，然后将得到的输出电压作为上标定点，再以传感器无负重时的输出作为下标定点，从而完成标定。尽管这样看起来有些粗糙，但是它很实用并且通常是最好的办法。

5. 噪声源

噪声通常会通过很多不同的信号源/电源进入称重传感器的输出信号中。交流噪声可以

通过接入传感器的激励电压线和电桥上的输出信号线耦合进入传感器。假设这些引线彼此十分靠近（理想情况下为双绞线），那么耦合噪声就是共模型的，因此大部分会被信号输入端的差分放大器和ADC测量的比例性所滤除。这一点十分重要，因为50Hz或者60Hz的电源噪声恰好处于被测信号的有效频率分量的中点。

如果电路自身的去噪特性无法完全滤除交流噪声，那么设计师就必须采用中心频率为50Hz或者60Hz的（这取决于电源频率）数字式窄带陷波器。尽管这会增加软件设计的复杂度，并且会衰减测量信号，但是根据应用及电气环境的需要，这种滤波器可能是必须的。

6. 故障条件

称重传感器可能出现以下的故障。

（1）传感器输入/输出引线断开。

（2）传感器中电桥元件损坏。

（3）激励电压源反接。

（4）输出电压反接。

所有上述故障都可以通过测量传感器各引脚间的电阻来检测出来，但是为了使系统能够自动完成这些工作则还需要额外的电路，并且会减少可用于检测称重传感器输出信号的ADC通道数，这是因为有些ADC通道被诊断电路占用了。在这些故障中，前3个属于“致命性”故障，一旦出现系统就无法工作。而一旦检测出第4种故障，还可以恢复（将输出电压反相即可）。 215

7.3 应用设计

本应用设计是一个十分简单的称重传感器器件，它能使用称重传感器精确测量0～50磅范围内的重物。为了简化标定过程，我们将采用已知称重电阻法进行标定，即接上一个阻值已知的电路作为上标定点。

7.3.1 系统指标

本系统采用Microchip公司的dsPICDEM v1.1板作为硬件平台，完成以下任务。

（1）测量0～50磅范围内的重物，测量分辨率为0.25磅。

（2）最大测量称重64磅。

（3）单称重通带采样速率为1kHz（假设被测信号的最大频率分量为200Hz）。

（4）对采样结果滤波，以去除60Hz的电源线噪声。

（5）允许用户通过RS-232串行端口（38.4kbit/s，8个数据位，1个停止位，无校验，无流控制）实现以下功能：

（a）标定传感器；

（b）指定测量上/下限值。

（6）通过串行端口每秒报告测量值1次。以文本形式报告称重。

（7）报告越界警报。若测量值小于下限则点亮演示板上的LED1，若测量值大于上限值则点亮LED2。

和温度监控系统一样，如果要将测量结果传输至其他电子系统而不是用户，那么最好采用速度更快的二进制数据协议。和热电偶系统实例一样，也有实例代码。 216

7.3.2　传感器信号调理

称重传感器的接口框图参见图7-2。它主要包括用于调整称重传感器输出信号的带有增益的仪表放大器，外部激励电压源（这里采用系统的5V电源），以及在ADC采样前用于限制放大信号频率的一对级联的二阶巴特沃思抗混叠滤波器。此外，该接口电路还包括一个继电器，它允许处理器切换一个阻值已知的标定电阻作为标定分路。

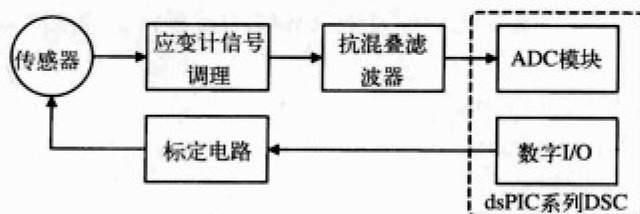


图7-2　称重传感器接口的框图（单通道）

请注意，这里我们可以采用和热电偶信号调理电路相同的差分放大电路（但是增益可能不同）和巴特沃思滤波器电路（但是这里采用不同的器件参数以实现更宽的通带）。实际电路的原理图参见图7-3。和温度测量应用实例一样，该原理图也是针对3.3V供电系统的，当使用dsPICDEM 1.1 通用开发板时，请将所有参考电压由–3.3V改为–5V。所有–3.3V改为–5V，所有1.65V改为2.5V。

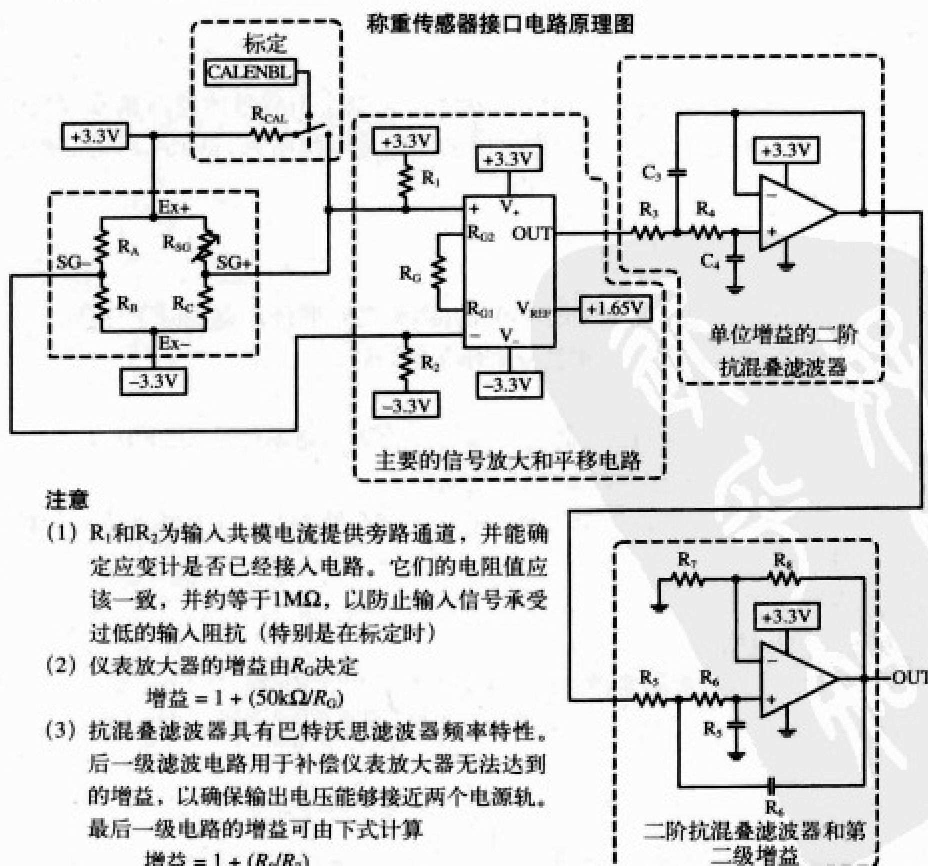


图7-3　称重传感器接口电路的原理图

7.3.3 数字滤波器分析

由于交流电源噪声位于被测信号的频带之内，因此需要使用陡峭的陷波滤波器将其滤除。这里所采用的滤波器的频率响应参见图7-4。

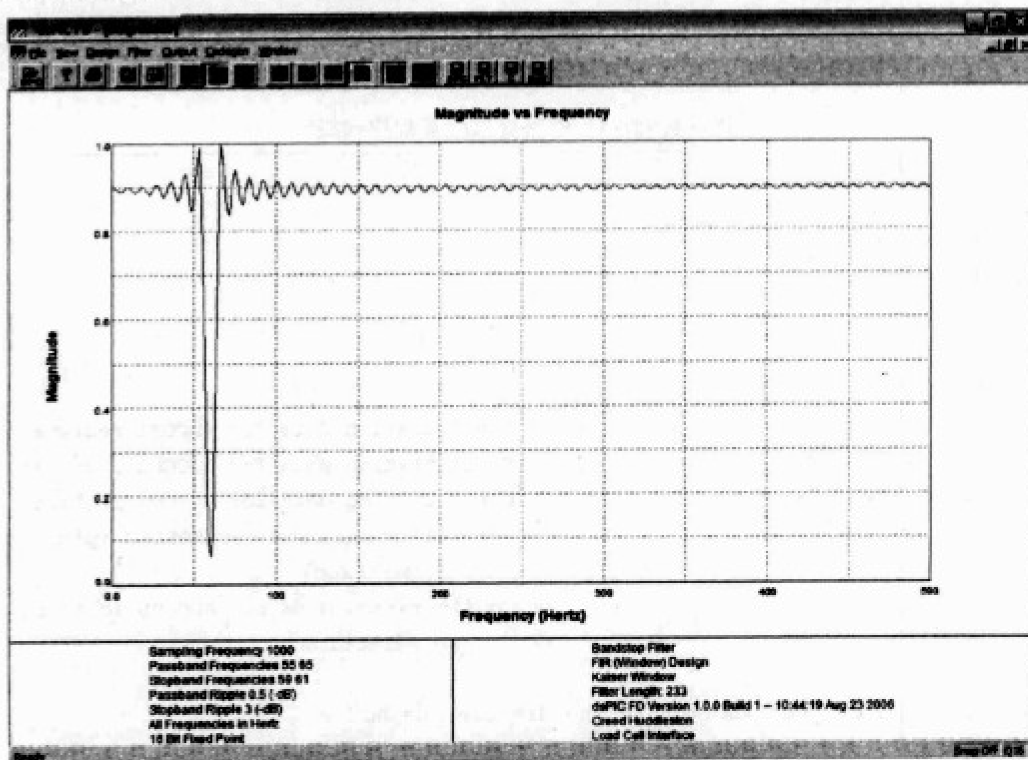


图7-4 用于滤除交流电源噪声的陷波滤波器的频率响应

7.3.4 数据分析算法

数据分析算法也十分简单，和温度监控系统采用的分析算法基本相同，只有很少的改动。该算法只是检查每个滤波后的采样值，并确定它们是否超出报警上限或下限，一旦发现越界，就点亮相应的LED以示警告。

217
~
218

7.4 固件实现

和温度监控系统一样，称重测量系统的固件实现也包括信号采样、滤波、数据分析以及错误处理，这些程序都基于Microchip公司的16-bit Peripheral Library和其DSP Library而设计，涵盖了大多数硬件接口的功能并提供了可靠且已测试的数字信号处理程序集实现滤波。

7.4.1 信号采样

dsPIC的片上ADC每毫秒采样一次模拟信号输入引脚AN7上的称重传感器信号。以触发ADC转换，使用的是自动采样模式和dsPIC的Timer 3。为了使本系统的分辨率最大，我们将假设输入信号是0～5V的单极性信号（即要求用户确保激励电压源不会被反接），因此ADC

[219] 输出可以采用无符号1Q15格式来表示。此外，为了减小快速采样的时间开销，将缓存区每写入8次采样结果后才产生1次中断。当然，还可以减小写入的次数，以加快系统的响应速度或者进一步减小与中断有关的处理器开销。

配置ADC采样参数的代码参见模块ADCIF.c中的函数ADCInit()。它调用了16-bit Peripheral Library中的OpenADC12()和OpenTimer3()例程，完成对ADC和Timer 3的配置，具体请参见ADCIF.c和Timer.c文件中的相关代码。

代码实例7-1 信号处理配置程序代码

(1) ADCIF.c中的代码

```
Uint8
    ADCInit(void)
    {
    //  Local Variables

    Uint16
        ui16ADCON1,                 // Configuration data for ADCON1 register
        ui16ADCON2,                 // Configuration data for ADCON2 register
        ui16ADCON3,                 // Configuration data for ADCON3 register
        ui16ADPinConfig,            // Configuration data for ADPCFG (pin
                                    //   configuration)
        ui16ScanSelect;             // Configuration data for ADCSSL (channel
                                    //   scan selection)

    //  Turn off the ADC module and disable the ADC
    //  interrupt to ensure that the configuration
    //  completes without interruption

    CloseADC12();

    //  Setup the ADCON1 register configuration
    //  data, of which the most important is the
    //  data format, the clock source, and the
    //  Auto Sampling mode

    ui16ADCON1 = ADC_MODULE_ON      & // Enable the ADC module
                 ADC_IDLE_STOP      & // Stop conversions during IDLE state
                 ADC_FORMAT_FRACT   & // Generate unsigned fractional data
                 ADC_CLK_TMR        & // Use Timer 3 to trigger conversions
                 ADC_AUTO_SAMPLING_ON;  // Enable the Auto Sampling mode

    //  Setup the ADCON2 register configuration
    //  data, which includes the ADC reference
    //  voltages, the number of samples between
    //  interrupts, and the buffer data order
    //  specification.
```

[220]

```
ui16ADCON2 = ADC_VREF_AVDD_AVSS    & // Use AVdd and AVss as ADC reference
             ADC_SCAN_ON           & // Enable channel scan mode
             ADC_SAMPLES_PER_INT_8 & // Gather 8 samples before interrupting
             ADC_ALT_BUF_OFF       & // Don't alternate the sample buffer
             ADC_ALT_INPUT_OFF;    & // Don't interleave the input

//  Setup the ADCON3 register configuration
//  data, which includes

ui16ADCON3 = ADC_SAMPLE_TIME_31   &  // Sample for 31 Tad
             ADC_CONV_CLK_SYSTEM &  // Use system clock for conversions
             ADC_CONV_CLK_32Tcy;    // Allow 32 Tcy for conversion

//  Setup the ADCSSL register configuration
//  data, which specifies which channels are to
//  be scanned by the ADC. Note that the way
//  the library is implemented, we specify the
//  ANx signals that we do NOT want to scan in
//  this list. In this case, we leave out AN7
//  since that's the only one we want to sample.

ui16ScanSelect = SKIP_SCAN_AN0  & SKIP_SCAN_AN1  & SKIP_SCAN_AN2  &
                 SKIP_SCAN_AN3  & SKIP_SCAN_AN4  & SKIP_SCAN_AN5  &
                 SKIP_SCAN_AN6  & SKIP_SCAN_AN8  & SKIP_SCAN_AN9  &
                 SKIP_SCAN_AN10 & SKIP_SCAN_AN11 & SKIP_SCAN_AN12 &
                 SKIP_SCAN_AN13 & SKIP_SCAN_AN14 & SKIP_SCAN_AN15;

//  Make sure that the pins associated with
//  the analog channel(s) we're using have
//  been configured as analog inputs. This
//  application uses AN7 as the signal input,
//  and the dsPICDEM board dedicates AN0 and
//  AN1 to the ICD2 interface, AN3 to the
//  digital potentiometer input, AN4-AN6 to
//  the analog potentiometer inputs, and AN8
//  to the temperature sensor input.

ui16ADPinConfig = ENABLE_AN3_ANA & ENABLE_AN4_ANA & ENABLE_AN5_ANA &
                  ENABLE_AN6_ANA & ENABLE_AN8_ANA;

//  Actually configure the ADC module. Note
//  that this will enable the ADC module, but
//  the associated interrupt must be enabled
//  by a call to EnableIntADC() after the rest
//  of the system has been initialized
```

221

```
    OpenADC12(ui16ADCON1, ui16ADCON2, ui16ADCON3, ui16ADPinConfig,
              ui16ScanSelect);

    //  Initialize Timer 3 to generate a 1 msec
    //  interrupt to trigger the ADC sampling

    Timer3Init(ADC_SAMPLE_RATE);

    return ST_OK;
    }
```

(2) Timer.c中的代码

```
void
    Timer3Init(Uint16 ui16SampleRate)
    {
    //  Local Variables

    Uint16
        ui16Period,                    // Timer period in counts
        ui16TimerCfg;                  // Timer module configuration data

    //  First, turn off Timer 3 and its associated
    //  interrupt so we can complete the initialization
    //  without being interrupted

    CloseTimer3();

    //  Compute the timer period in counts based
    //  on the instruction clock frequency. The
    //  instruction clock frequency is computed
    //  by multiplying the crystal frequency FOSC
    //  by the phase-locked loop scaler PLL and
    //  then dividing by 4 since each instruction
    //  takes four system clock cycles.

    ui16Period = (Uint16)(((FOSC * PLL / 4) / ui16SampleRate) + 1);

    //  Configure the timer itself

    ui16TimerCfg = T3_ON            &    // Enable Timer 3
                   T3_IDLE_STOP     &    // Turn off Timer 3 in IDLE state
                   T3_GATE_OFF      &    // Not gating the timer
                   T3_PS_1_1        &    // Set the prescaler to 1:1
                   T3_SOURCE_INT;        // Use internal clock for timer
```

```
OpenTimer3(ui16TimerCfg, ui16Period);   // Configure the timer

return;
}
```

中断服务程序是ADCISR()，它也位于ADCIF.c模块中。中断服务程序会清除ADC中断标志。如果未能清除该标志，会导致处理器在执行完该中断服务程序后再次进入中断服务程序，从而使应用程序陷入中断服务程序的死循环。

代码实例7-2 ADC中断服务程序代码

```
ADCIF.c中的代码:
      void __attribute__((__interrupt__))
         ADCISR(void)
         {
         //  Local Variables

         fractional volatile
              *pfrADCBuff,             // Pointer to ADC data buffer
              *pfrInputSignal;         // Pointer to global input signal data
                                       //   buffer

         Uint8
              ui8DataCount;            // ADC data index

         //  Clear the ADC interrupt flag and

         IFS0bits.ADIF = 0;            // Clear the ADC interrupt flag

         //  Copy the A/D conversion results to the buffer

         //  g_frSensorSignal[]. Note that the samples
         //  from each channel are contiguous, so they will
         //  have to be placed into separate signal arrays
         //  the samples are to be filtered. We don't do
         //  that here in order to minimize the time spent
         //  in the ISR.

         pfrADCBuff      = &ADCBUF0;         // Point to the first entry in the ADC
                                             //   data buffer
         pfrInputSignal = g_frInputSignal;  // Point to the first entry in the input
                                            //   signal data buffer

         for (ui8DataCount = 0; ui8DataCount < ADC_SAMPLE_COUNT; ui8DataCount++)
              *pfrInputSignal++ = *pfrADCBuff++;  // Copy the next ADC sample to the
                                                  //  global input signal buffer

         //  Set the Filter Event to signal the main
```

223

```
        //  processing loop that we have new data to
        //  filter

        g_vui16SysEvent |= EVT_FILTER;
        }
```

7.4.2　数字滤波器实现

数字滤波器实现与第6章中介绍的温度传感器所用的数字滤波器类似，唯一的区别是滤波器系数不同。这里采用了FIR滤波器，它要求应用程序创建并初始化一个名为FIRStruct的数据结构，以保存滤波器的相关状态变量。对新数据的滤波处理只需调用DSP 库例程FIR()并传入新的采样值即可。

在本应用中，我们将采用带阻滤波器，其相关参数如下：采样频率是1000Hz，通带频率是55Hz和65Hz，阻带频率是59Hz和61Hz，通带纹波是0.5dB，阻带纹波是3dB，滤波器类型是Kaiser。

224

将上述参数输入dsPIC Filter Design软件并产生滤波器代码（保存在SensorFilter.s中），就会得到我们所需的滤波器数据结构。请注意，该滤波器的长度比热电偶应用所设计的滤波器长得多（本例为233个抽头，而前例只有51个抽头），这是因为这里要求的滤波器具有更快的滚降特性。

7.4.3　数据分析算法实现

数据分析算法是由Analysis.c模块中的AnalyzeData()函数实现的。该函数极为简单，只是反复将滤波后的数据与参考阈值做比较，一旦发现采样值超出限定值，就点亮上限或下限警报指示灯LED。

代码实例7-3　数据分析代码

```
Analysis.c中的代码：
Uint16
    AnalyzeData(void)
    {
    //  Local Variables

    float
        fSensorValue;                   // Current scaled sensor value

    PSensorCfg
        psenSensorCfg;                  // Pointer to sensor configuration

    Uint16
        ui16SensorIndex;                // 0-based sensor index

    //  Check the sensors one at a time to see
    //  whether the sensors' current values
```

```
    //  exceed any enabled alarm limits

    psenSensorCfg = g_senSensorCfg;       // Point to first sensor configuration

    for (ui16SensorIndex = 0; ui16SensorIndex < MAX_CAL_SENSORS;
         ui16SensorIndex++, psenSensorCfg++)
        {
        //  Convert the current filtered sensor
        //  reading into its corresponding parameter
        //  data value

        fSensorValue = ScaleData(g_frFilteredSensor[ui16SensorIndex],
                                 psenSensorCfg->fGain,
                                 psenSensorCfg->fOffset + 0.0005);
        g_fSensorValue[ui16SensorIndex] = fSensorValue;

        //  Are we checking the lower alarm limit
        //  and, if so, is the sensor's filtered
        //  data value less than the lower limit?

        if ((psenSensorCfg->ui16Flags & SENFLG_ENBL_LOW_ALM) &&
            (fSensorValue < psenSensorCfg->fAlarmLevel[ALARM_LOWER]))
            psenSensorCfg->ui16Flags |= SENFLG_LOW_ALM_ST;
                                    // Alarm limit exceeded
                                    //   so set its flag

        //  Are we checking the upper alarm limit
        //  and, if so, is the sensor's filtered
        //  data value greater than the upper limit?

        if ((psenSensorCfg->ui16Flags & SENFLG_ENBL_HI_ALM) &&
            (fSensorValue > psenSensorCfg->fAlarmLevel[ALARM_UPPER]))
            psenSensorCfg->ui16Flags |= SENFLG_HI_ALM_ST;
                                    // Alarm limit exceeded
                                    //   so set its flag
        }

    return ST_OK;
    }
```

7.4.4 错误处理的实现

本应用的错误处理可以十分简单，也可以十分复杂，这主要看系统是否需要测量电桥中每个引脚的电阻。为此需要将称重传感器电桥的每个引脚都接入独立的模拟输入通道，这会显著增加电路的复杂度，并会损失单片dsPIC系列DSC利用处理器内部资源可以监测的称重

传感器通道数量。

错误处理代码位于Analysis.c文件的CheckForErrors()函数中，它包括能根据所测电压接近地电平或者接近满幅电平来分别判定电桥连接是短路或是开路。如果检测到任何一种情况，那么代码就会设置一个标志使报警LED（LED2）闪烁。直到故障被清除，应用程序才会熄灭报警LED。

226

7.5 小结

至此，我们已经开发了两套应用系统，它们都使用了dsPIC的ADC和数字滤波器。在接下来的第三个应用中我们将使用完全不同的方法：它将通过定时/计数器对外部时钟输入计数，从而确定管道中液体的流量。

227

第8章 流量传感器

下面要讨论的流行的传感器是流量传感器，它能测量通过系统的物质（通常是液体）的流量。尽管有关流量的定义方式很多，包括*质量流量*、*体积流量*、*层流量*以及*紊流量*，但是本质上都是要测量出为实现特定目的而流动的物质的数量[①]。因此，系统设计师通常对*质量流量*（单位时间流过系统某点的物质的质量）更感兴趣，但如果流动的是液体，并且液体的密度为常数（至少在通过系统中感兴趣的某点或者某些点时，其密度为常数），那么设计师就能在不影响检测系统性能的条件下，通过测量*体积流量*（单位时间通过的体积）来代替质量流量。事实上，很多情况下都是这么做的，本章设计实例也不例外。

现实中流量测量主要应用于哪些领域呢？对于大多数人来说，最常见并且最普通的例子就是电表和水表，它们分别用于测量消耗的电量或水量。还有很多系统，比如输油管道以及新生儿重症监护室内用于监测病人呼吸的小型系统，也都采用了流量传感器，用来监测和控制关键的工作参数。毫不夸张地讲，商业贸易也依赖于可靠的流量检测系统，以确保能够准确地向世界各地发送数以千计的各种燃料、食品或其他商品。

8.1 流量传感器的类型

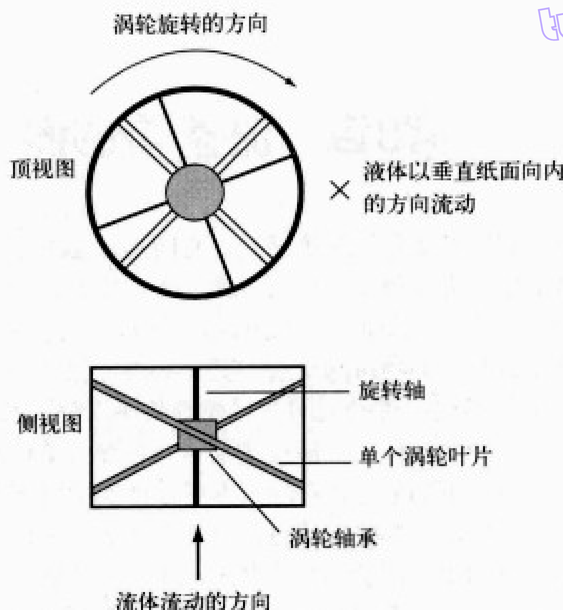
我们要讨论的是两种常见流量传感器：一个是涡轮传感器，用于测量体积流量；另一个是*重力传感器*，用于测量质量流量。之所以要重点介绍涡轮传感器，主要有两点原因：一个是在线的涡轮传感器应用很广泛，因此设计师有必要了解它们，第二点可能也是更重要的是，涡轮传感器的输出形式与前面介绍的两种传感器都不一样，这是一种所谓*基于频率*的输出信号，信号频率会随流量的增加而增大。我们将会看到，这种新型的输出信号既有优点也有缺点，而掌握如何使用基于频率的传感器也是设计师的必备技能。

8.1.1 涡轮传感器

涡轮传感器通常用于测量液体或者气体，它看起来就像一个风扇或者螺旋桨，其中，叶片转轴的方向对应流动的方向，并且风扇的叶片会覆盖流体通道的整个横截面，具体参见图8-1。叶片的布局很关键，它不仅可以使流过的液体给传感器施加尽可能大的力（这对于低流速的情况非常有用），而且还能确保流体以正常压力通过传感器，从而防止在流量较大时压坏涡轮。

由于叶片的转轴方向与流体流动的方向平行，因此很难直接测量轴的旋转方向。为此，涡轮传感器通常含有窗口，用于从侧面观察叶片。通过对叶片在单位周期时间内划过窗口中固定的某点的次数计数，并测得流体物质的密度以及流动路径的横截面积，传感器就能确定通过传感器的液体的容积。

① *Transducer Interfacing Handbook: A Guide to Analog Signal Conditioning*, Edited by Daniel H. Copyright 1980, 1981 by Analog Devices, Inc., p.24。



230

图8-1　涡轮流量传感器的例子

由于叶片和系统的其他部分不相连，因此乍一看，检测叶片的转动是一件使人望而生畏的工作。幸好，在很多情况下只需要安装一个光源以一定的频率为窗口照明就可以做到，这样当叶片经过照明点时就会反射出一定频率的光，通过测量反射光信号就可以得到一组反复上下跳变的输出信号。图8-2就是这种反射信号的例子，通过计算该信号在单位时间内的尖峰个数，就能知道叶片旋转得有多快。在8.2.4节我们还将深入研究这个问题。

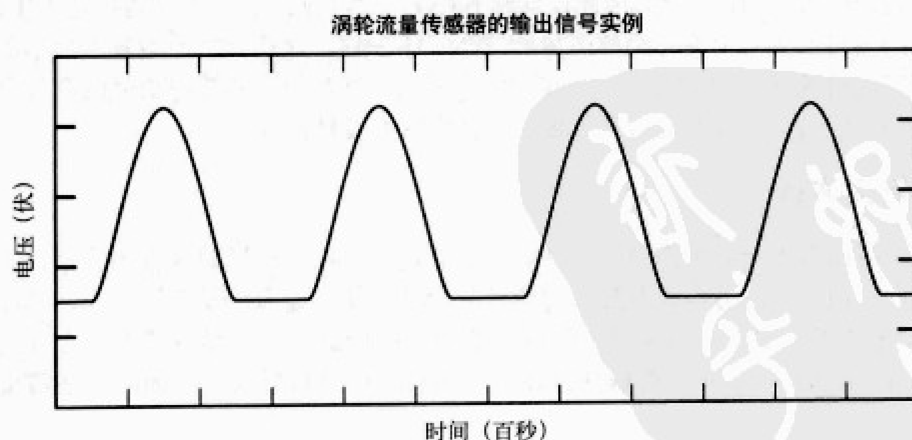


图8-2　涡轮流量传感器输出信号的实例

温度传感器和载重传感器是全电子型传感器，而优秀的涡轮流量传感器设计与此不同，同时需要高水平的电气技术及机械技术。尽管这里将重点研究传感器的电气接口，但是读者一定要注意流量的机械方面其实也很重要，而且还应理解加入传感器后对被监测系统的影响问题。

231

8.1.2 重力传感器

涡流流量传感器用于测量在给定时间内通过某个横截面的物质的容积，与此不同的是，重力传感器则是测量给定时间内通过某个横截面的物质的质量。尽管这有时比容积流量测量更准确，但是重力流量传感器比涡轮流量传感器贵得多（贵几十倍），这限制了它们只能应用于需要极高精度并且能承受昂贵成本的场合，比如标定实验室。

8.2 流量测量的要点

流量测量的很多要点适用于所有的传感器系统，比如测量的范围和分辨率，但是，还有些是流量传感器特有的，比如在低流速和高流速环境下面临的挑战。

8.2.1 测量范围

测量范围与被监测物质的物理特性、用于测量的涡轮的特性以及流动物质所在系统的特性有关。在某些流体表面，被观测的物质可能因为剪应力（shear force）撕裂材料而遭破坏。此外，由于磨损材料在高流速下会快速地摩擦叶片，因此物质的特性还会严重影响传感器本身，特别是叶片的寿命。叶片被磨损后面积有所减小，流过损失了的传感器面积的流量就无法检测，因此会严重影响传感器的测量精度。叶片面积减小还会降低传感器在低流速下的测量能力（因为传感器上的旋转压力减小）。

涡轮的特性决定可测量的流量上下限。这是因为，当流量太小时就无法使涡轮转动（流体的压力不足以克服叶片沿轴旋转时的摩擦力），而当流量太大时又会使叶片变形，从而产生测量误差。另一方面，若叶片转轴发生磨损，就会出现叶片“抖动”问题，这也会使测量精度降低。后一种问题非常普遍，因此我们将在8.2.4节中讨论，并在该节中专门列出一小节“噪声源”。

232

最后，流量测量所需范围由系统本身决定。当要求的流量测量的等级超出所用传感器的能力时，可以用已知大小的涡轮“采样”部分流量，通过测量穿过涡轮的流量就能换算出实际的流量。

综上所述，有些典型的涡轮流量传感器系列的测量范围达0.2～60加仑[①]/分钟（gallon per minute，GPM）[②]。此外，特制的传感器还能测量超过该范围的更大的流量。

8.2.2 测量分辨率

前面两种传感器都能直接产生与感兴趣的参数对应的电压信号，而涡轮流量传感器与之不同，它产生信号的频率（而不是电压）与被测参数对应。为了获取这种信号，我们需要采用一种略微不同的模拟预处理电路，将该模拟反射信号转换为数字式的频率信号，并且用该数字信号驱动dsPIC系列DSC的一个定时/计数模块。而定时/计数模块能够在另一个具有系统时钟周期分辨率（dsPICDEM v1.1评估板配置为33.9ns）的高速定时器的配合下，在其时钟输入信号上对前述频率信号的变化次数计数，这样就能满足任何实际涡轮流量传感器的要求，并得出正确的读数。

① 1加仑约等于3.8升。——编者注

② CITO Products 公司的RotoFlow™生产线。

8.2.3 测量精度

涡轮流量传感器能够在满量程下达到2%甚至更高的精度，并且，只要涡轮旋转通道畅通且涡轮本身不会因磨损或者老化而损坏，那么测量结果就具有很好的可重复性。通常，测量的精度在极限值附近有所下降（即在极低流量或者极高流量条件下）。如果传感器的涡轮组件发生严重磨损，特别是涡轮旋转轴方向上的轴承发生磨损，传感器的测量精度会显著下降。幸运的是，通过采用一些方法就能够在一定程度上改善这种精度损失，至少可以使系统性能缓慢地下降。通常这也给用户提供了一种监测系统的机会，并能提醒用户安排必要的维护以避免因为忽视而造成故障。

233

8.2.4 测量的挑战

涡轮流量传感器在流量处于上下限的条件下工作时面临很多严重问题。流量过低所面临的问题是，流体的压力不足以克服叶片与旋转轴之间的摩擦力，因而无法使叶片旋转。在这种情况下，流体就会渗透过叶片而叶片却不旋转。于是，尽管传感器的读数显示流量为零，可是实际的流量却不为零。为了补偿这种情况，可以在涡轮中增加叶片的数量，这样就能产生更大的旋转力，从而降低可测流量的下限。但是这样做又会增大传感器输出信号的频率（这是因为叶片越多，每旋转一周发生的变化就越多），从而会增加数据捕获的难度。

然而，在信号频谱的另一端，当流量过大时，信号波峰和波谷的幅度会显著减小，这使得反射信号变得较为平坦。因此要求比较电路能够动态调整比较器的阈值，以便能正确地将模拟反射信号转换成数字信号。同时，系统固件必须能够监测模拟信号的极值以便设置合适的阈值。

随着使用时间的增加，固定传感器叶片的轴承会磨损轴的固定孔，使得孔径增大。孔径增大后就会使叶片在轴向上摇晃，忽上忽下。于是，叶片在连续转动过程中会斜着通过参考光照点，从而导致发射光尖峰在“有效”期内反复变换。根据传感器所用材料以及流量大小的不同，轴承磨损问题可能很快就凸现出来，一旦出现问题就会造成严重后果。

1. 信号的特征

模拟反射信号的幅度相对较大，通常可达数百毫伏至数伏，因此很容易放大它。但是考虑到传感器信号的负载能力以及克服共模噪声的需求，推荐采用仪表放大器对该信号缓冲后再进行进一步的处理。

涡轮信号的频率上限很容易计算，它等于叶片个数乘以传感器允许的最大转速。因此为了满足奈奎斯特准则，最小采样频率等于信号最高频率的两倍。在本实例中请注意，我们采样的目的只是为了确定信号的最大和最小值，而不必从所采样的电压信号中提取参数信息。提取参数值的任务是由计数器完成的，而设计计数器时主要考虑的是确保计数器在数据采样周期内不会溢出。

234

2. 材料密度补偿

为了将流量－容积测量问题转换为质量—容积测量问题，传感器必须考虑材料的密度。密度的大小与材料有关，将它与容积—流量测量的结果相乘就能得到对应的质量流量，当然这里假设被测物的密度在被测范围内为常数。如果不满足该假设，那么就需要以分段线性化的方式用密度补偿系数进行校正，这和我们在热电偶测量中使用的方法类似。

3. 线性化

在大多数情况下，不需要采用特别的线性化算法，这是因为信号频率通常是随流量增大而线性增加的。但是，线性度在流速的上下限附近略有下降，因此有时需要采用三点分段线性化方法以保证传感器在测量上下限附近的精度。

4. 标定

涡轮流量传感器的标定要求相当低，这是因为传感器输出信号的特性是由传感器的机械尺寸决定的，而与电气响应无关。一般来说，对于给定型号的传感器来说，其标定曲线是固定的，并且在现场无需标定。

5. 噪声源

有三种主要噪声源会使涡轮流量传感器的信号质量下降：轴承摩擦、流体浑浊以及传感器接口中模拟部分的电子噪声。前面已经详细讨论了轴承摩擦的起因和影响，而流体浑浊的主要影响是会显著减少穿过涡轮叶片的反射信号，从而不利于传感器系统识别叶片通过参考区时的有效变化。

尽管在前两个应用中未讲述电子噪声，但是在这里它也是个问题。它主要影响比较器的输入值，输入值中的噪声信号导致比较器错误地改变输出值，就会使流量测量值偏高或偏低。

我们还将在8.3.2节中探讨解决这三种噪声源造成的问题的方法，但是有一个要求：这三种噪声源必须都能够得到满意处理。

6. 故障条件

涡轮流量传感器能够检测出的唯一的硬件故障是传感器引脚断开，而且这也得在一定条件下才能办到的。其他的系统问题，比如过流，虽然也可以检测出来，但是它通常被视为有效的状态（尽管它们可能已经超出了传感器的测量范围），这将在介绍数据分析时讨论，这里就不做介绍。

检测传感器引脚断开的最佳方法是在传感器输出端对地接一个敏感电阻，并且周期性地检测该敏感电阻两端的电压，确保它始终与期望的传感器输出电压基本一致。根据涡轮传感器本身输出电压的幅度，在敏感电阻和ADC输入端之间还应接一个高输入阻抗的放大器。

8.3 应用设计

和前面开发的应用系统一样，这里开发的流量传感器系统也十分简单，但是它涉及流量测量的很多重要方面。我们设计的系统将采用涡轮型流量传感器，测量范围是0.2GPM（大约相当于0.8升/分钟，记为0.8LPM）至8GMP。当测量值高于测量上限值或者低于测量下限值时，对应的越界报警LED就会被点亮。当信号线断裂或未连接时，硬件故障LED则会闪烁。

8.3.1 系统指标

本系统采用Microchip公司的dsPICDEM v1.1板作为硬件平台，并实现以下功能。

（1）流量测量范围为0.2～8GPM，分辨率为0.01GPM。

（2）允许用户通过RS-232串行端口实现以下操作（串行端口相关参数为38.4kbit/s、1个停止位、无校验并且无流量控制）：

（a）设定各种传感器或流体的流动因子。

(b) 设定测量范围上下限值。

236

(3) 通过串行端口每秒钟报告一次测得的流量值。流量报告采用文本格式。

(4) 报告越界报警信息。当测得的流量值小于下限值时，显示检测值为“- - - -”，并点亮评估板上的LED1；当测得的流量值大于上限值时，显示检测值为“+ + + +”，并点亮LED2。

(5) 报告硬件错误（传感器引脚断开），并使LED3闪烁。

与前两个应用系统一样，如果检测信息是发往系统的其他电子部件而不是用户时，那么最好选用传输速度更快的二进制数据传输协议，这可以参考示例软件。

8.3.2 传感器信号调理

由于传感器信号的固有特性各有不同，本实例所采用的信号调理电路与前面两个应用略有不同。特别是，传感器输出信号经缓冲后被送往比较器并产生相应的二进制数字信号，以便dsPIC系列DSC的外部计数器测量。为了改善电气噪声的影响，可以采用滞后比较器，这种比较器的阈值可调，从而使得dsPIC系列DSC能够补偿高流量下的低电平信号。图8-3是本应用所采用的模拟信号调理电路的框图。

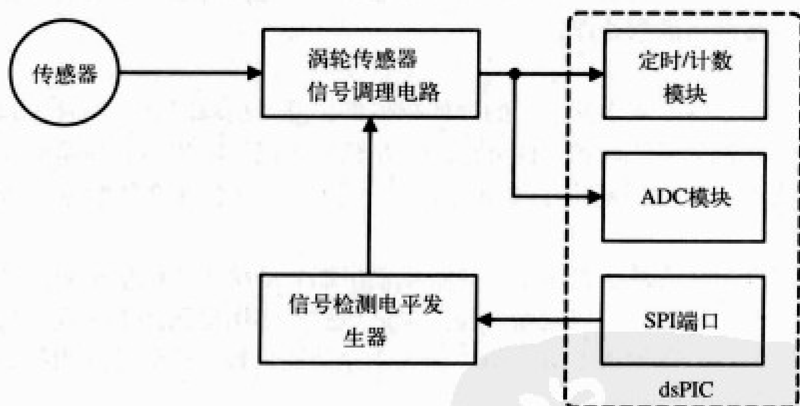


图8-3 流量传感器的信号调理电路的框图

8.3.3 数字滤波分析

本系统先采用低通滤波器平滑来自传感器的原始电压信号，然后分析信号以确定最小值和最大值。与前两个应用系统不同的是，这里不需要陷波滤波器来消除电源线噪声，因为我们只要为比较电路增加滞后环节，就能克服电源线噪声的影响。尽管采用基于频率的传感器信号增加了系统的复杂度，但是这也给系统带来了好处，克服电源线噪声就是其中一点。

237

显然，累计一段时间内划过参考点的叶片个数本身就具有一定的平均功能，因此它能有效去除由轴承摩擦产生的干扰信号。尽管每个叶片穿过参考点的周期长度会因为叶片摇摆而略有不同，但是通过计算一段时间内穿过的叶片总数后，这种周期波动就会被平均化。我们可以通过延长计算更长时间内叶片变化个数来进一步改善上述平均效果，但是这样做会牺牲系统响应速度。当然也允许用户自定义计数的时间长度，但是由于用户通常并不熟悉滤波的细微差别，因此可能只好采用有限的几个设定值。图8-4所示为“正常”的流量信号，其中

每个叶片变化尖峰的间隔基本上为常数，而图8-5所示的是同一个传感器在使用一段时间后发生磨损情况下的检测信号。请注意，尽管第二幅图中峰间时间与第一幅图相比变化很大，但是这两个信号的平均峰间时间却基本相等。

由此，读者可能会问我们为什么要使用dsPIC系列DSC定时器模块的计数功能，而不是dsPIC系列DSC的输入捕获功能，来提取叶片变化的次数并计算出平均变化次数（捕获单元只能测量两个信号变化之间的时间）。主要原因是，如此精细的分辨率会使测量结果中出现很多毛刺，并且当轴承磨损后这种毛刺会愈加严重。此外，在高流速条件下，由于此时每个叶片变化之间的时间非常短，因此基于每个叶片变化时间来计算流速会花费大量处理带宽。通过收集大量的变化然后取其平均，比计算大量流速值再平均能得到更好的结果。

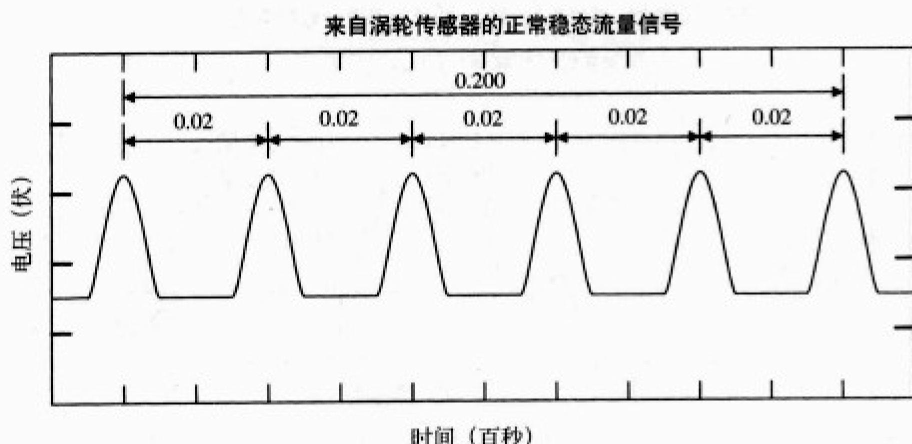


图8-4 叶片无摆动时正常的流量信号

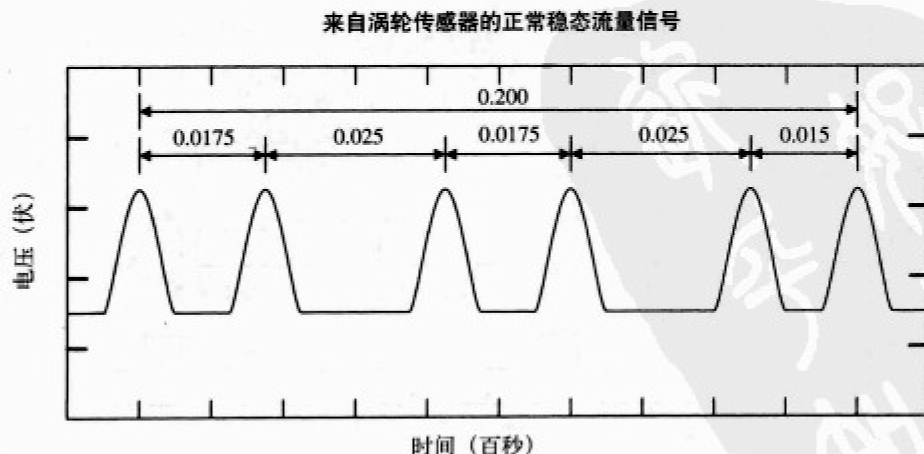


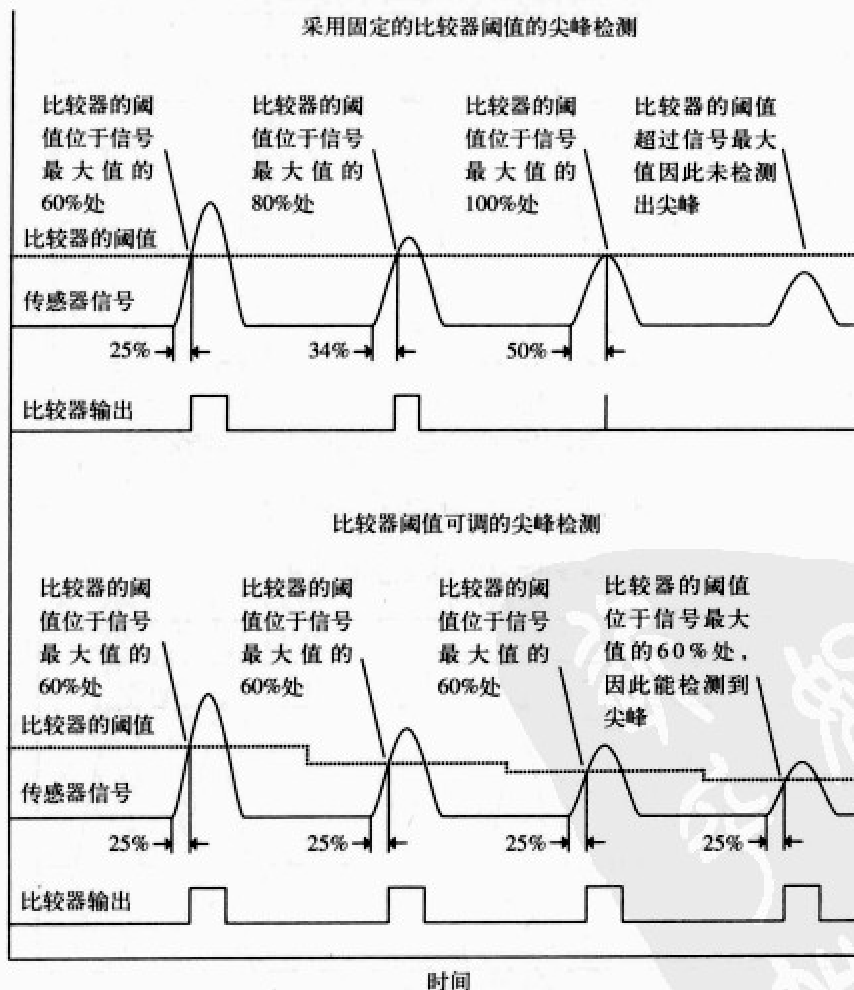
图8-5 叶片摆动时流量信号的例子

8.3.4 数据分析算法

流量传感器的数据分析算法与前两章介绍的温度检测以及称重检测应用相比略为复杂。和那两个应用一样，流量测量系统也需要将其滤波输出值与上、下限报警值比较，除此之外，它还需要周期性地读取来自涡轮传感器的原始模拟信号，以确定如何调整尖峰检测比较电路的偏置电压。请读者回忆一下，原始信号的电压会随着叶片变化速度的增加（即流速提高）而下降。理想情况下，即使传感器输出发生变化，处理器也能将比较电压值设定为与传感器输出信号相同的相对值，从而使峰间（interpeak）时间不会随传感器输出电压的变化而发生漂移。图8-6所示就是我们希望能够避免的情况。

239

比较器阈值大小对检测到的尖峰的位置及有效性的影响



除非比较器的阈值被调节成始终是传感器输出信号的固定百分比，否则尖峰检测比较器的输出信号前沿会随流量而变化（这是因为尖峰的高度会随频率而变）。最坏的情况可能是无法检测到有效尖峰，从而导致报告的流速值远小于比实际流速值

请注意，在实际中，我们事先知道适用于第一个尖峰的正确的比较器阈值与适用于之前尖峰的阈值完全不同，因此，第一个尖峰检测的时机可能并不是最佳的。但是，通过将整组的检测尖峰平均后就没什么影响了

图8-6 错误的峰间周期变化的例子

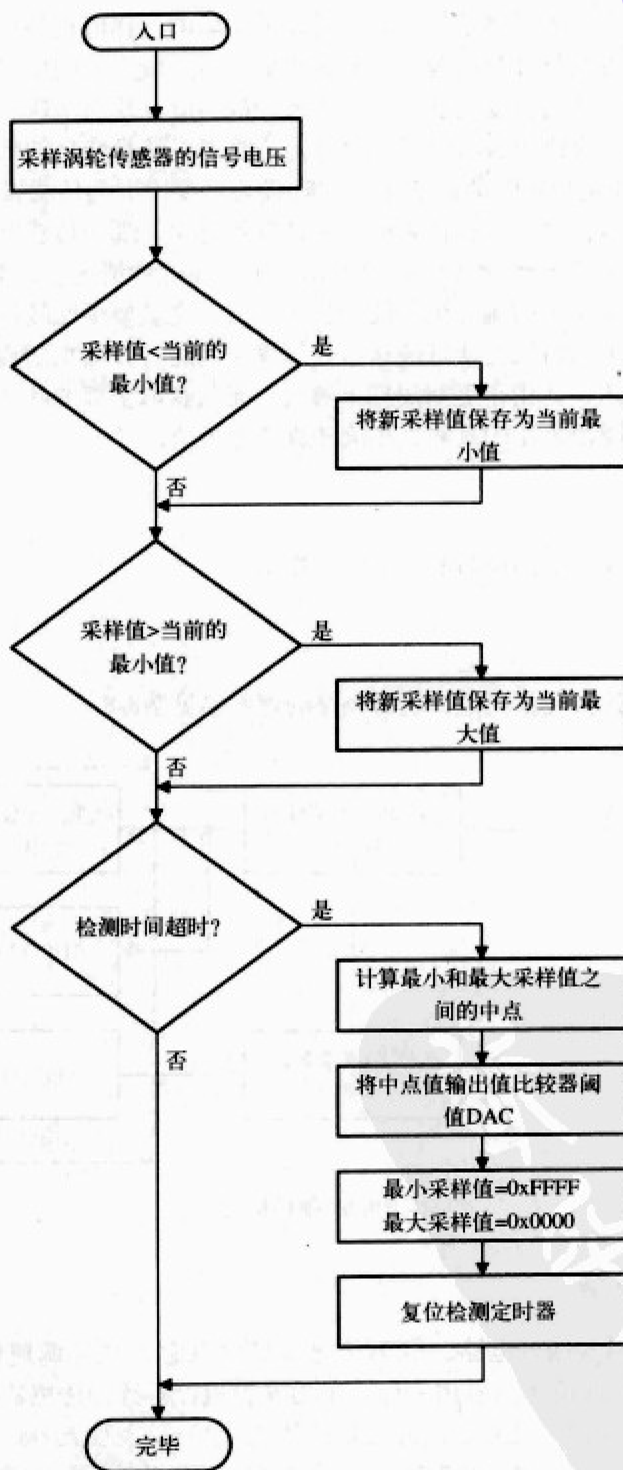


图8-7 比较器阈值电压计算算法

检测原始传感器输出以及设定比较器阈值的算法参见图8-7。它先采样传感器信号，接着对其略微滤波以净化信号，然后检测采样值是否超出预定的极限值。当算法发现某个采样

值小于当前的最小值或者大于当前的最大值时，就用新的采样值代替原来的极值。一旦获得足够的采样值，那么算法就计算所检测到的最小值与最大值之间的中点值，并输出该中点值作为新的比较器阈值，最后再复位已采得的最大和最小值，并启动新一轮采样。

在确定多久调整一次比较器阈值时，设计师应当考虑流量变化的最大速率（即流速波动有多快）。这在从低流量条件变为高流量条件时极为重要，因为传感器输出将从相对的大幅度信号变为小幅度信号。如果比较器阈值调整速度不够快，那么传感器输出的最大值就可能低于比较器的阈值，从而导致比较器的输出无变化。在这种情况下，dsPIC系列DSC看起来好像未发现流动，可事实上的确发生了流动！乍一看，这就要求比较器阈值更新的速度尽可能快，但是更新速度太快时又会使低流速条件下发生问题，因为更新周期可能比叶片变化的时间还短。在这种情况下，由于更新周期不够长，无法获取全部的叶片变化，因此检测到的极值就无法反应出传感器输出的真实最小值和真实最大值。

240~242

8.3.5 通信协议

本应用采用的基本通信协议与前两个应用相同。

8.4 硬件实现

流量传感器的硬件实现很简单，所需电路的框图参见图8-8。

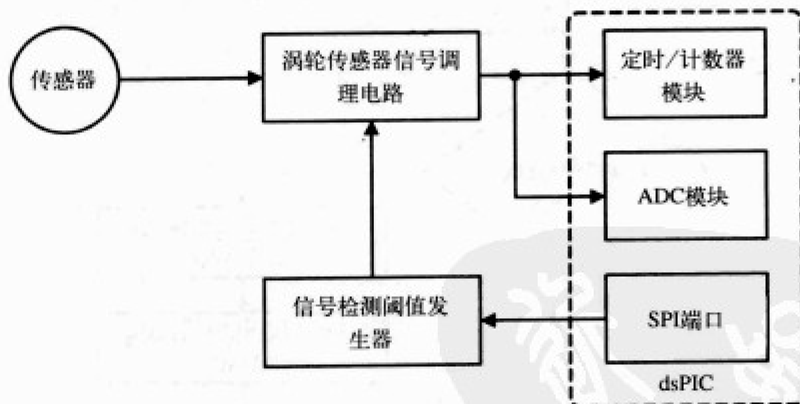
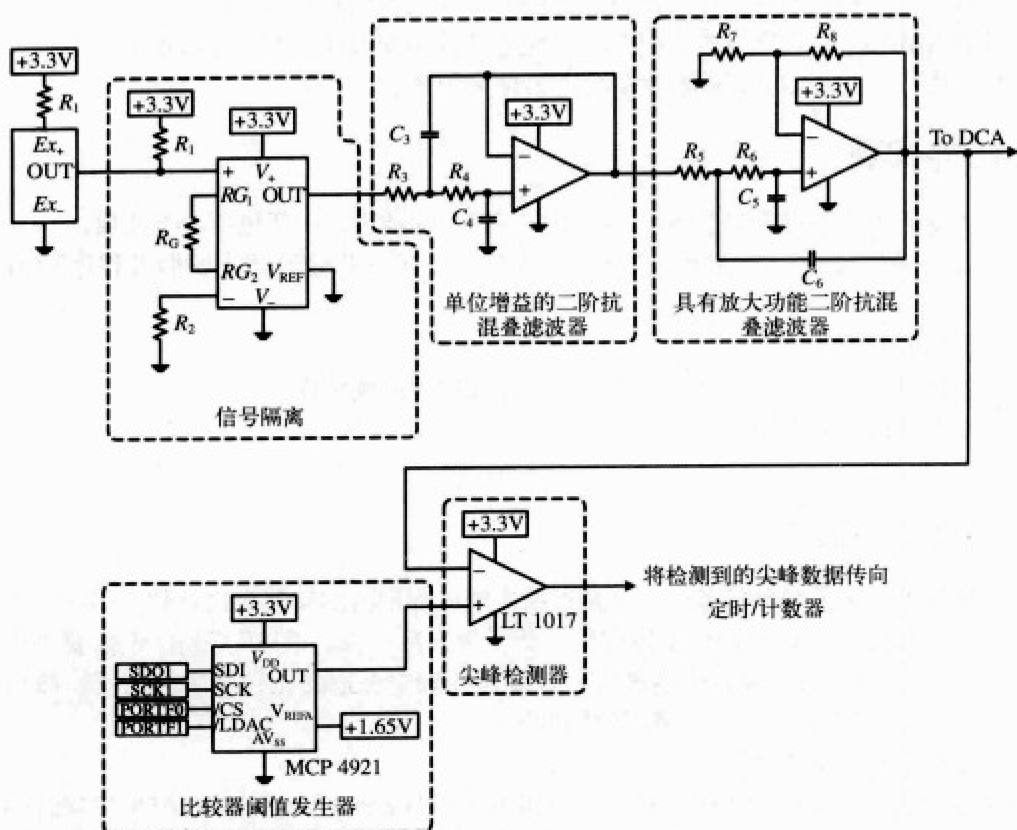


图8-8 硬件框图

涡轮传感器接口电路

图8-9是涡轮传感器接口电路，它为传感器提供电源，并且调理传感器输出信号以便dsPIC系列DSC处理。和前面的应用一样，采用仪表放大器缓冲传感器输出信号，但是由于涡轮传感器输出电压的幅度足够，因此这里的增益为1；仪表放大器还作为传感器和系统其他部分之间的缓冲器。仪表放大器的输出信号再经过二阶巴特沃思滤波器处理后，被同时送往比较器和ADC通道，前者将产生数字频率信号以便我们做后续分析，后者则用于检测传感器信号电压。请注意，和其他两个应用实例一样，这里所示的原理图是针对3.3V系统，因此必须经过适当的电压变化后才能应用于5V的dsPICDEM 1.1通用开发板。



注意

(1) R_1和R_2为输入共模电流提供了旁路通道，并且还能用于检测传感器是否接入电路。这两个电阻的阻值约为100kΩ，以避免输入信号承受的输入阻抗太小

(2) 仪表放大器的增益可由R_G控制，关系式为：

$Gain = 1 + (50k\Omega / R_G)$

(3) 抗混叠滤波器具有巴特沃思滤波器的频率响应。最后一级电路的增益能够补偿仪表放大器的增益不足，从而使最终的输出信号接近两个电源轨。最后一级电路的增益方程为：

$Gain = 1 + (R_8 / R_7)$

图8-9 涡轮流量传感器的接口电路

为了实现比较器阈值可调，原理图采用了Microchip 4921数–模转换器（DAC），该器件的功能与ADC恰好相反，它能将数字输入量转换为模拟输出电压。与ADC一样，DAC也是比例的，其输出量等于输出电压乘以输入的数字量与最大可写入数字量之比。对于工作于5V的12位的DAC来说，这就意味着其输出电压范围可达0～4.9988V，分别对应于0～0x0FFF（0～4095）。尽管12位DAC足以满足本应用的要求，但是我们还可以改用14位（16 384个等级）甚至16位（65 536个等级）的DAC以获得更高的分辨率。由于dsPIC系列DSC可以通过SPI通道与DAC接口，因此增加DAC的分辨率并不会增加额外的硬件，只需稍稍修改一下写入DAC的命令字即可。当采样dsPICDEM 1.1通用开发板时，为了进一步简化电路设计，示例软件实际上采用了板上SPI数字电位计来设定比较器的阈值。

243～244

由于我们需要检测施加在比较器上的传感器信号，因此抗混叠滤波器必须保证在感兴趣的频带内具有单位增益响应。由于巴特沃思滤波器在通带内具有最佳的平坦增益响应，并能在超出最高有效频率点后迅速进入阻带，因此它是理想的选择，利用它我们能够有效去除不期望的频率成分，并且不会影响有用的传感器信号频谱。

8.5 固件实现

流速测量的固件采用了与前面两个应用实例相同的结构，主要包括以下几项。

（1）数据采样模块，包括叶片变化累加程序、ADC采样定时器中断服务程序和ADC中断服务程序。

（2）数据滤波模块。

（3）数据分析模块，包括叶片计数分析和比较器阈值调整程序。

（4）硬件错误检测模块。

245

（5）通信模块。

8.5.1 数据采样模块

数据采样模块负责两项任务。一个是对给定时间周期内的叶片变化计数，另一个是调整比较器阈值电压以便最优地检测上述变化。为了做到后一点，dsPIC系列DSC需要采样送往比较器输入端的经过预处理的传感器信号等级，并确定合适的比较器阈值电压，然后通过SPI端口输出至数字电位计以产生模拟阈值电压。

1. 传感器信号电平检测器

和温度检测及称重检测系统一样，本应用也采用Timer 3产生1kHz的ADC采样时钟。由于我们只想获得给定时间周期内传感器信号的极值，因此将ADC配置为采样8次才产生1次中断。由于流量传感器数据是单极性的，因此采样数据采用无符号的小数形式表示。当产生ADC中断时，中断服务程序（ISR）就会将采样值送入g_frSensorSignal滤波器的缓冲器中，并且设定EVT_FILTER事件，从而使主处理进程循环对传感器数据滤波并更新DAC的输出电压。

2. 叶片变化计数器

在本应用中，dsPIC系列DSC的定时器模块的外部时钟输入引脚与比较器的输出端口相接，因此，定时器模块能对定时器外部时钟输入脚的上升沿次数计数。再通过第二个定时器产生精确的时间基准，就能确定出给定时间内叶片变化的次数，从而得到对应的流量值。

本应用采用Timer 5构成一个16位同步计数器，这样，我们能够测量在一个相对较长的时间内的叶片变化次数。同时，这也扩展了可以测量的流量的下限。数据采样模块采用Timer 3产生500ms的时间基准，用于累加叶片变化次数，这只需简单地在Timer 3的1ms时间中断中读取Timer 5的累加计数值并在其累加至500次时对其复位即可。在作者使用过的流量传感器中（4片或者6片的），通常都是累计500ms内叶片变化次数，效果都不错，但是最佳的时间周期应取决于特定应用要求的响应速度以及预计的流速。500ms的累计次数对应的数据更新速度是每秒2次（500ms更新1次），这也不一定适合于给定的系统。如果要求响应速

246

度更快，那么累计时间就要缩短（这通常会更新阈值），但是这样做又会降低算法本身的平

均滤波效果，并且，为了检测出非零的流量，就必须在每个累加周期内检测出至少1次叶片变化，因此会增大系统的最小可检测流速。

当500ms定时器计时到达时，Timer 3的中断服务程序（ISR）就会将Timer 5的当前16位值（它对应叶片变化的次数）存入全局变量g_ui16SensorCount，并且设置EVT_ANALYZE时间有效，从而通知主处理循环调用数据分析软件计算流速。

8.5.2 数据滤波模块

前两个例子的传感器信号将被滤波以改善参数分析能力，而本应用的滤波则是为了获取信号电压更好的读数，以便它能最优地用于设定比较器的阈值。

传感器信号滤波

传感器信号滤波器其实是一种低通滤波器。和称重检测应用中所采用的滤波器电路不同，本系统不打算采用陷波滤波器来去除耦合进传感器信号中的电源线噪声，因为对于本应用来说，即使信号混有这种电源线噪声也没关系。如果去除50Hz或60Hz附近的窄带频率信号，会导致流量信号出现“死区”，在死区内比较器输出不会发生改变。由于电源线噪声对比较器信号的影响非常小，因此我们可以放心地忽略来自电源线的耦合噪声。

尽管滤波器的采样速率只有1kHz，但是，由于本系统只需要检测一个通道，并且不必逐样本地处理数字化数据，因此处理任务不是很重。尽可能地减少数字化处理的开销，就可以腾出更多的时间进行滤波，因此我们可以实现一个具有极其平坦通道增益的滤波器，具体参见图8-10。该滤波器具有59个抽头、最小四项余弦窗，正如读者看到的，它的通道增益等于1。

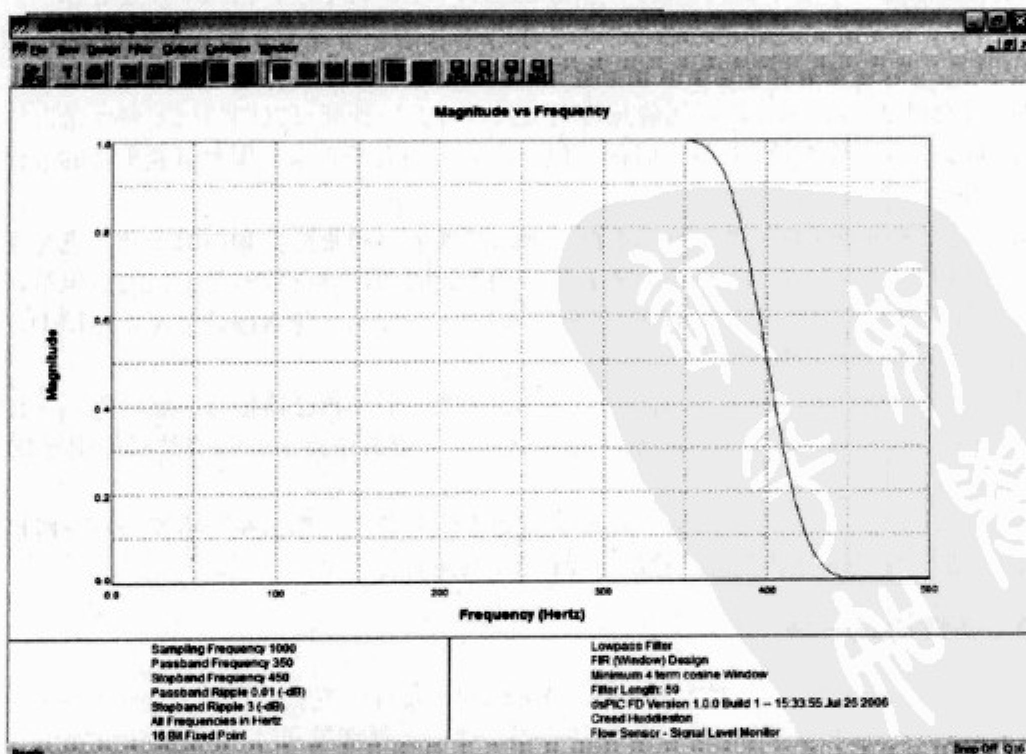


图8-10 信号监测滤波器的响应

8.5.3 数据分析模块

数据分析模块负责以下两项任务。

（1）比较最新滤波后的流速值与用户指定的报警界限值，并且合理地设定对应的越界报警指示LED的显示状态。

247

（2）分析缓冲后的传感器输出信号电压，以确定：

（a）传感器是否真的连接到系统；

（b）如果已经连接了传感器，那么就计算最佳的比较器阈值电压，并将计算结果输出至DAC。

1. 流速分析

流速值的数据分析算法与前两个系统的相同。只需将最新的滤波后的流速值与指定的报警界限值比较，并据此正确地设定对应的报警LED的状态，然后将全局计数器增加1以记录距离上次通过通信端口报告检测数据经过了多少时间。如果距上次报告数据经过了1s，那么函数将设定EVT_REPORT_RESULTS事件，以通知主处理循环应当通过通信端口报告最新的流速值了。最后，该程序将清除EVT_ANALYZE_DATA事件，以表示对当前滤波后的流速值已完成数据分析处理并返回。

248

2. 信号大小分析

传感器信号大小分析首先要检查传感器是否已连入系统输入端。如果传感器未正确连入，那么仪表放大器同相输入端的上拉电阻就会迫使仪表放大器的输出为接近正电源轨电压的常数。而在正常情况下，该信号大多数时候都应当低于电源轨电压，仅在叶片通过照明点时短时间达到接近电源轨的电压（这是因为此时通过叶片的反射信号最强）。为了确定传感器已经连入系统，本应用只需简单地检查滤波后的传感器信号的大小，如果该信号超出预设值很长时间，比如说0.5s，那么就表明传感器未接入系统。设计师可以调整触发硬件错误检测所需的精确的最小检测电压以及持续时间，但是它们必须大于正常工作时可能出现的最大信号电压以及该电压持续的时间。

如果前述分析表明传感器未接入系统，那么就置位全局变量g_bHWError，点亮硬件故障指示灯LED，并跳过比较器阈值分析程序，因为此时的传感器信号是无效的。但是，如果传感器已连入系统，那么本程序就应清零g_bHWError变量，熄灭硬件故障指示LED，并接着执行比较器阈值分析程序。

比较器阈值分析也很简单，只需逐样本地检测滤波后的传感器信号，然后确定在当前阈值更新周期内的最大和最小值，最后计算出这两个极值的中点值，并将结果输出至DAC以设定新的比较器阈值。

一旦完成比较器阈值分析（或者，如果传感器信号丢失，那么在传感器检测分析后就立即结束），那么主处理循环就会清除EVT_ANALYZE事件。

8.5.4 通信协议模块

主处理循环一旦接收到EVT_REPORT_RESULTS事件，它就会调用FormatResults-Msg()函数。本程序将格式化最新的流量数据，然后主循环就通过TransmitResults()例程将格式化的信息发送至主机。一旦发出结果消息，那么主处理循环就复位EVT_REPO-RT_RESULTS事件，以表明该事件已经处理完毕。这里所采用的协议与温度检测及称重检测

系统的相同，因此就不做进一步讨论。 249

8.6 小结

本章开发的流量表应用系统为我们展示了一种新的数据采集方式，传感器输出信号的频率（而非电压）包含了我们感兴趣的参数信息。这是一种简单而有效的传感器数据编码方式，但是在使用它时有一些注意事项。设计师必须仔细保证系统能够处理最小和最大的有效信号频率，特别要注意极高频速率信号，因为性能较差的系统很可能无法处理它们。显然，我们可以通过增加对流速敏感的叶片数量（这可以使得在低流速条件下的累计时间更长，从而扩展可测量的下限，并且在高流速条件下计算间隔时间更短，从而提高传感器的灵敏度）或者通过增大到传感器电源线的敏感电阻（这可以增加诊断能力）来改进系统性能，但是本章的重点是引导读者掌握基本的传感器接口方法，而不是研究提高各种性能。

在最后一章，我们将从技术和商业角度研究智能传感器市场的发展方向。 250

第9章 智能传感器发展趋势

至此，读者已经相当熟悉智能传感器以及实现它们所需的概念了，而本章我们要将注意力转向这种强大器件的未来发展趋势。与任何新兴技术一样，如今的智能传感器也蕴涵了巨大潜力，但是为了使它们的应用更加广泛仍需要解决大量问题。如何才能预测该领域在未来几年的发展呢？是什么能将智能传感器推向市场？又是什么会阻碍它的应用呢？我们可以预测出智能传感器将扩展至哪些市场吗？它们又会有哪些新功能？本章将研究这些问题。尽管我们不可能完全解答这些问题，但是至少可以让读者掌握智能传感器发展历程中的重要里程碑，并能为读者提供深入研究这些主题的资源。

9.1 技术发展趋势

智能传感器包括三个基本部分：敏感元件、运算元件以及通信接口。它们所涉及的基本技术发展很快。在未来几年，我们将目睹智能传感器性能的迅猛发展，它们的功能和互联性能会显著增强，同时系统成本又会显著降低。下面就详细介绍这三个方面的发展情况。

253

9.1.1 敏感元件的发展趋势

有人可能会认为传感器领域的发展极为缓慢，但这种看法是完全错误的。该领域的发展使得很多传感器的灵敏度明显增加，特别是在化学分析领域。随着灵敏度的提高，设计师就能创建出尺寸更小的系统，它们的功耗更低，并且为获得精确读数所需的分析物质也更少。

这一点在微射流领域体现得最为明显，它专门用于控制和分析极微量的液体。在这种领域中，被测液体的体积极小，通常是微升甚至是纳升级①。该概念已被应用于喷墨打印机以及便携式血液检验设备等场合，因此微射流应用领域很广泛。其中有一个子类，被称为数字微射流，它可以精确操作一滴液体的一部分，从而可以用于化验液体的化学组成或者执行其他有用的任务。

正如我们想象的那样，可靠处理微量物质的能力取决于系统准确测量液滴的重要参数的能力。特别是，系统能否精确地测量和控制物质及其必经通道的压力和温度就显得尤为关键。如果无法保证合适的条件，就会严重影响微射流系统的性能。但是，如果具备所需的环境，那么就会获得很大的收获：可以使必须测试的物质数量减少数个量级，于是只需几分钟就能报告检测结果（而不像以前需要几天时间）；此外，还能提高系统的便携性并降低成本。尽管微射流不是体现传感器技术显著提高的唯一领域，但却是表现较突出的领域。

设计师还能够预见到，未来传感器会朝着小型化和灵敏度更高的方向发展。与此相补充的领域（比如纳米技术）的发展会拓展特种传感器的应用范围，并促进研发出新型传感器。然而，为了有效施展各种传感器的优势，设计师不仅要掌握传感器技术的最新发展动态，而且还要学习产生传感器功能的基本物理现象。

254

① 微升等于百万分之一升（10^{-6}升），纳升等于十亿分之一升（10^{-9}升）。实际上，它们比一滴液体还少。

9.1.2 运算元件的发展趋势

到目前为止，大部分人都听说过摩尔定律，它指出运算能力大约每两年就会提高1倍①。尽管我们可以预见到未来的运算能力还会提高，但是对于嵌入式微处理器系统（比如我们已采用过的dsPIC系统），除了运算原始数据外还有其他方面需要考虑。功耗、物理尺寸以及开放的硬件和软件架构都是重要因素。在某些应用中，它们可能比处理能力更重要。

1. 超低功耗

在很多情况下，系统功耗是延长便携式或者远程应用中电池寿命的关键因素，并且减小功耗还能降低恼人的自发热问题。包括dsPIC系列DSC以及大多数处理器在内的很多数字器件的功耗以及发热都与它们的时钟速率成正比。时钟速率翻倍在使系统的运行速度翻倍的同时，也会使芯片的功耗翻倍。尽管这是个缺点，但是它也为我们提供了一种降低功耗的方法。当芯片的某部分无需工作时，可以关闭它们或者在这些情况下降低芯片的时钟频率，这样就能显著地降低功耗。这对于只采用电池供电的系统极为重要，它可以使系统在一套电池供电下就能工作数周、数月甚至数年。

我们已经看到处理器以及传感器信号的调理电路的功耗还将显著降低，并且能预见到未来这种趋势的发展会更快，因为移动设备要求电池使用寿命越来越长（要求能够测量数年，而不是几天）。特别是Microchip公司在一个夸张的宣传活动中宣称该公司已经研制出了由葡萄柚供电的基于微处理器的温度检测系统，以展示他们的元件多么节能②，而这种低功耗工作方式在未来将变成正常工作方式，而不像现在还只能在特殊情况下才能达到的。

遗憾的是，任何事情都有代价。为了降低系统功耗，芯片设计师不得不将芯片供电电压降低到3.3V，甚至更低。这反过来会严重影响系统，特别是模拟信号链的性能。降低电源电压后，会显著增加模拟信号链相对电源电压跨度噪声。比如，如果在电源电压为5V时，模拟信号的噪声为10mV，那么噪声等级就等于2%（10mV/5V＝0.02）。但是同样的10mV噪声在电源电压跨距为2.5V时就会产生4%的噪声等级。如果处理不当，就会导致严重错误。这种影响在激励电压源不得不在电气噪声环境下连线很长时还会更为严重。事实上，很多老式的传感器系统采用±10V甚至±15V传感器激励电压源的原因之一就是，大跨度电压可以减小输出信号的噪声影响。

255

降低工作电压的另一个负面影响是，它们难以和老式的5V甚至更高工作电压的系统相接。尽管不是绝对做不到，但是将新式的低电压系统与老式的高电压系统相连时，通常需要可靠的电平转换电路。尽管新型器件的I/O端口通常能够承受5V电压，但是低电压器件的输出信号却可能不足以驱动5V系统的输入，产生逻辑高信号。这个问题会随着越来越多的系统采用低电压供电而淡化，但是在目前阶段的设计中还必须考虑它。

2. 尺寸不断缩小

智能传感器系统在功耗需求降低的同时，系统的尺寸还会更加精简。目前，小型化、电池供电的传感器系统已经占据了美国传感器市场的四分之一，但是未来的系统还会更小。所

① 摩尔定律是以Gordon Moore的名字命名，他是英特尔公司PC Processor部的创始人之一，并于1965年做出了这个预言。事实上，根据英特尔公司的网站介绍，摩尔定律实际上说的是芯片上的三极管数量每两年就会增加一倍。由于芯片上的三极管个数与其处理能力近似成正比，因此该“定律”以一种更加流行的形式流传开来。有关摩尔定律的详细内容，请参阅http://www.intel.com/technology/silicon/mooreslaw。

② 展示品采用了Microchip公司制造的采用nanoWatt™技术的PIC单片机。

谓的微尘传感器（mote），是一种完全自给的设备，具有板上的无线电连接，从而可以构成ad hoc（自组织分组无线网络）通信网络（在9.1.3节会详细介绍），并与网络内的其他微尘传感器或者主机交换数据。同样令人振奋的消息还有，目前已经开发出长度约为5mm的微尘传感器，并且最终目标是使系统的体积仅为$1mm^3$。[①]这么小的封装，其尺寸相当于一粒沙子的大小，但是却包含了智能传感器所需的所有部件：敏感元件、处理器、无线电以及电池。

这种小型化带来了很多封装和操作上的问题，特别是难以在如此微小的空间内放置一个能维持系统长期工作的能源。同时，与这样微小的设备通信也是个重要问题，因为我们难以将如此小型的设备与电缆连接。下一节我们将讨论通信的问题，涉及微尘传感器之间的通信以及它们与更大的系统的通信问题。 256

9.1.3 通信技术的发展趋势

电路技术的发展对智能传感器未来的影响一样重要，它对通信技术的促进作用简直令人难以置信，而通信技术的发展又会推动智能传感器应用的显著增长。特别是微尘型和其他的小型传感器，它们将采用无线通信方式，构成ad hoc传感器网络，这种网络可以自动组网、无人干涉（或者只需少量人工），并能快速展开。大尺寸的传感器系统尽管仍采用有线通信连接，但是它们更多地采用高速（100Mbit/s甚至更快）网络标准，以便于收集、散播和存储系统产生的大容量信息。下面我们将看到智能传感器通信未来发展的具体方向。

1. 无处不在的因特网

Harbor Research公司是一家战略咨询和研究机构，其研究内容也涵盖智能传感器，它们用“无处不在的因特网”这个词描述普遍计算和连在因特网上的广泛分布的设备的结合[②]。根据目前的发展趋势，这是一种恰当的描述。支持访问因特网的各种设备（比如手机、PDA、笔记本电脑等）不断涌现。智能传感器与因特网的连接技术也随之同步发展。虽然可能有人认为该方法是依赖于“电气设备”的，但是事实上很多应用都会从这种通信提供的控制中收益。比如，用户可以设定汽车发动机的参数，从而可以在家庭旅游时和拖船时提供更高的功率，或者对新的学徒驾驶员限制最高时速。尽管这些在目前还未商业化，但是它们在技术上是可行的，我们只需等待那些具有敏锐市场眼光的公司来开发它们。

然而，在这些互联的系统成为现实之前，还有很多问题必须强调。第一个问题很实际，必须为所有设备设定网络地址。目前的因特网协议是所谓的IPv4，它最多支持4.3×10^9个（43亿个）不同的地址，这听起来好像很多，但是全球平均下来，每人不足一个。由于每个人的工作和生活中需要使用很多工具（汽车、器械、电脑以及非电子设备），而单位（如学校、公司、政府等）则需要管理更多的设备，因此目前的地址分配方案肯定无法将所有设备连接起来。幸运的是，下一代因特网协议——IPv6，支持多达3.4×10^{38}个地址。这样就能轻松地满足未来的需要[③]。当然还可以采用其他的方法，但是它们都没有IPv6成熟。 257

---

① 加利佛尼亚大学伯克利分校电子工程系的Kristofer Pister教授 于2004年3月23日出版的网络版*ComputerWorld*杂志中发表的文章，具体参见http://www.computer-world.com /mobiletopics/story/0,10801,79572,00.html。

② Harbor研究机构的网站上（www.harborresearch.com）有很多对遍布的因特网中的商机深入研究的白皮书，以及利用这些机会的具体策略。

③ 仔细研究IPv6就会发现，这种新协议能够为目前全球约65亿人提供多达50×10^{27}个IP地址。有趣的是，IPv6的想法也来源于PARC。

第二个问题是带宽。如何才能有效地传输这些互联系统产生的大量信息呢？尽管我们已经看到因特网连接的带宽正在迅速提高（线缆、DSL等）。如果按比例地想象，未来的通信要求比现在更高的传输容量。Internet2项目[①]正在开发运行速度达10Gbit/s（10^9bit/s）的更快的因特网基础设备，但是直到2006年秋也只有美国有这样的设备。在高速路由和信息筛选方面还需要做大量工作，通过减小不必要的网络交换，还能使网络的可靠性更高、成本更低。

2. 无线通信

如果所有新连接的系统都需要采用电缆通信，那种情况简直无法现象，它的花费极高并且笨拙得无法使用。幸运的是，无线通信的应用正变得更加广泛、更易实现，并且成本更低，在很多情况下使得系统可以"剪断连线"。针对不同的应用，这可能是指仅消除通信电缆，但是它渐渐拓展到采用电池供电或者被动供电设备中，从而还将剪断电源线。

无线通信的方案有很多种，它们适用于各种不同的应用系统。新近出现的ZigBee™协议就特别适合于低功耗、低数据传输速率的应用，它可以在单电池供电条件下工作数年。在这些应用中，通信的间隔和频率都有所限制，以便降低功耗。而其他协议，比如流行的用于无线局域网（LAN）的802.11x协议，则支持连续通信，并且可以用于像实时监视和控制的应用中（当然还有为临时中断所做的准备）。ZigBee和802.11x协议在地理范围上都有所限制。而另一个极端的情况则是无线通信，它采用了蜂窝调制解调器或者其他能够连接全球设备的技术。我们可以预见到，无线通信技术的应用会愈加宽广，它的基础设备成本会更低，安装更方便，并且可用性更好。

258

3. 安全性

下面需要特别关注的是如何保证互联设备的网络不被非法监视甚至窜改。根据系统报告的信息类型，这个问题对于智能传感器系统来说和商业机构一样重要。特别是，必须防止用于控制生产的敏感信号出现讹误或被"窥探"，从而防止外人破坏生产或者收集所有人的信息。这就要求确保有线和无线网络的安全。

协议设计师现在认识到安全对稳健通信的重要性，并且最近开发的协议（比如ZigBee）在开始设计时就将安全作为其中的基本组成部分，而不是在设计之后又附加上的。遗憾的是，附加的安全手段会增加信息比特流的处理开销，降低数据吞吐量，并且根据具体实现，还会增加系统的软/硬件成本。然而，最终，安全性遭破坏时造成的损失通常都会超过产品的实际成本。

4. ad hoc 网络

在很多网络应用中，时间和金钱花销最大的就是维护网络。维护任何大规模网络需要高水平的网络管理员花费大量精力（和成本），因此，有时很多面向智能传感器的应用可能无法承受。为了使传感器网络兴旺发达，就必须使相对技术水平较低的用户能够快速熟悉和掌握网络，并且还要求网络节点能够自动将自己配置成合适的工作方式。这种网络就是所谓的*ad hoc*或者称为*自组织网络*，并且我们看到第一个可用的这种系统已经出现。ZigBee就是一个这种网络的例子，它允许节点无需大量配置就能接入或者离开网络，并且还有其他的实现方式（尽管它们可能是受专利保护的）。我们可以预见到在不久的未来（就是接下来几年）这个领域定会有重大进展。

259

在了解智能传感器领域的技术发展趋势后，下面将介绍该领域的经济发展趋势。

① 有关Internet2项目的更多信息，请参阅该组织的网站www.internet2.edu。

9.2 经济方面的发展趋势

有三点经济发展会推动智能传感器在未来被采纳，它们是：人口老龄化、生产的日益全球化以及新商机的出现。这三个趋势会强有力地增加对智能传感器系统的需求。

9.2.1 人口老龄化

在所有的发达国家中，人口出生率都已经低于维持人口数量所需的标准，并且这样的状况已经持续很长时间了。这将导致人口老龄化。尽管人们很早就已经意识到了人口尤其是劳动力的老龄化问题，但是由于美国婴儿潮时期出生的一代人以及遍布全球的二战后出生的一代人的退休，使得老龄化的影响现在才开始显露。在工业、医药以及其他行业，我们都将很快缺少大量的劳动力，并且眼下没有足够的人立刻接替他们[①]。如果我们要使经济能够保持以现在的水平以提供商品和服务，那么单个工人的生产效率就必须成倍提高，才能弥补由于退休工人引起的生产力下降。实现这一目标的最有效、也可能是唯一的方法是，使用能够自组织、自诊断并且比现在需要更少人工干预的智能传感器。

顺便说一句，不仅是发达国家有这样的问题。目前全球大多数国家的人口出生率都在下降，这当然也包括发展中国家。除非人口出生率增加（这还真有可能实现，但是需要时间），否则所有国家都得面临同样的窘境。

9.2.2 生产的日益全球化

无论你如何看待全球化的问题，它都已经成为事实，由于公司都愿意使用世界各地最有效的资源，因此可以预见到，生产全球化在未来仍将继续。尽管全球化可以降低公司的运营成本，但是它也使得公司难以管理分散的团队，收集或者发布他们需要的信息以便做出有效的决策。为了弥补这一缺陷，公司会使用更多的智能传感器系统，以便为决策制订者提供实时信息，并提供技术支持以确保组织正常运营。尽管这些决策制订者和技术人员可能不（事实上往往不）在同一个地点为某项活动提供支持，但是智能传感器系统的互联性可以确保他们能够很好地完成工作。 260

和人口老龄化问题一样，全球化问题也不仅仅是发达国家面临的困难。一些发展中国家通常被看作外包生产的接受者，这些国家的公司现在也发现必须与国内和自己距离很远的公司甚至是国外的外包公司合作，才能保证成本的竞争力。

新的商机

19世纪末期，美国西部的铁路网显著减小了运送商品和人口的时间和成本，这为有事业心的组织提供了新的商机。与此相同，智能传感器的衍生领域也为具有洞察力的公司提供了新的途径，来获得分布式技术能力和全球联网的设备所带来的利润。领先的公司已经在很多应用中采用智能传感器为客户带来收益，并以收费的形式从客户那里返回一部分收益。Harbor研究机构的一份有趣的研究报告[②]显示，大约有一百个应用领域可以从网络化的智能设备中受益。

---

① 这并不是说年轻一代的工人无用，只是说年轻的工人无法接替所有的退休工人。

② 参见*Venue Segmentation Map for Intelligent Device Networking and Management*，可以从www.harborresearch.com网站下载。

智能传感器系统中，传感器生产商和集成商现在有能力通过与监测相关的服务以及使用传感器创造出持续的收益，而不是像已往那样依赖于传统的单元销售（unit-sale）方式。正如第1章提到的那样，这可以为最终用户带来很多好处，同时为生产商带来稳定的收入，并能加强供应商和买家的联系。简而言之，购买者通过商品—采购方式增加主导，这一方面为供应商提供了区分它们的产品的方法，另一方面又能避免消费者感觉到供货商的提供产品和服务是人人相同的。

9.3 小结

本书的基本假设是智能传感器是大规模并且不断增长的传感器市场的未来主导产品，而dsPIC系列DSC为构建各种传感器提供了极好的技术基础，因此，有些功能尚未实现的原因可能主要是受限于设计师的想象力。尽管智能传感器领域要求掌握很多领域的知识（包括模拟信号调理、数字信号处理以及少量数字设计的知识[①]），但是它提供了一个平台，任何拥有创造力的设计师都能做出不朽的设计。

261

最后，正如艾伦·凯[②]的名言所说，“预测未来最好的办法就是创造它”。读者要有勇气更加深入地探索智能传感器领域，并且将他的独到见解应用于某个或者某些应用。不但设计智能传感器是一项智力挑战，而且研制每个新型传感器都增强了我们了解世界和改造世界的能力。如果某个系统能广泛地用于生产全球数以十亿计的产品，或者通过研制一个灵巧关节以便更好地了解世界、改变人民的生活，那么就会得到丰厚的回报。如果读者能通过自己的敏锐和洞察力来应用本书所讲述的知识，那么就会使他设计的智能传感器发生巨大变化。

262

① 这么说不是要诋毁固态数字设计的重要性。只是说以前需要的很多个独立的数字电路现在都集成到一枚dsPIC芯片里。为了使用dsPIC系列DSC，仍然需要有效地掌握数字设计技术，但是由于系统集成度的提高，对数字设计技术的要求没有以前那么高了。

② Alan kay，2003年图灵奖得主。

附录A 本书附带资源

附带资源[1]上的软件包括因特网资源的网页链接以及本书开发的三个应用系统的源代码和工程文件。为了浏览网页，请使用Windows Explorer™（文件管理程序，请不要与网页浏览器Internet Explorer混淆）找到根目录的文件index.htm，然后双击该文件。这样就会启动网页浏览器并载入网站的首页。

A.1 网站资源

附带资源中的网站链接为读者提供了一种简单获取与本书所讨论内容有关资源的途径，特别是，可以获取参考文献以及联系提供有关器件和设备的供应商。由于每个参考资料网页上的链接都连接到另一个网站，因此有些连接可能会过期，甚至不存在。如果发生这种情况，请读者使用链接中的短语在因特网上搜索，如果该网站仍存在，那么就能找到它。 265

A.2 三个应用系统的源代码

本书开发的三个应用系统的源代码都是基于Microchip C编译器开发的，完整的源代码以及相关的工程文件位于下列不同的目录中：

- \Source\Thermocouple——热电偶传感器系统
- \Source\Load Cell——称重传感器系统
- \Source\Flow Meter——流量表传感器系统

这三个应用系统的软件都需要在Microchip MPLAB v7.40集成开发环境（IDE）下使用Microchip C编译器v2.02（也可使用更高版本，但是不能使用更低的版本）编译和链接后方可运行。读者可以从Microchip公司的网站（www.microchip.com）免费获得MPLAB IDE以及学生版的C编译器。

如果读者希望修改应用系统中所使用的数字滤波器，那么就需要购买dsPIC Filter Design™软件，它也可以从Microchip公司的网站上获得。为了运行这些应用程序，读者还需要一块Microchip dsPICDEM 1.1通用开发板以及ICD2在线调试器，以便能将程序文件下载到dsPICDEM板上。这些设备也可以在Microchip公司的网站上获取。 266

[1] 本书附带资源请登录图灵网站（www.turingbook.com）免费注册下载。——编者注

附录B dsPIC系列DSC的初始化以及系统启动代码

当dsPIC系列DSC复位并初次启动时，它首先运行中断向量地址（00000H）单元内的代码，此时所有中断都是关闭的。为了正常工作，该代码必须首先跳过中断向量表以及备用中断向量表，而直接进入应用代码空间，并且开始执行所谓的*启动代码*。启动代码是固件的关键组成部分，它主要负责配置C语言运行环境，以便应用程序（即便不是全部，那么大部分应用程序也是用C语言编写）能够正常运行。在调用C代码前，第一步必须调用启动代码，否则执行任何C程序都会出错，并且通常很难诊断出错误原因。

由于启动代码是负责初始化C运行环境的，因此它与所用的编译器有关，并且通常作为编译器的一部分一同发布。Microchip编译器提供两个启动代码文件①，一个是`crt0.o`，它能初始化所有由程序存储器读取的初始化数据，并将所有非初始化数据清零；另一个是`crt1.o`，它并不初始化数据。除此之外，这两个模块都完成以下动作。

（1）将栈指针寄存器（`W15`）和栈指针限制寄存器（SPLIM）初始化成链接器指定的数值。

（2）如果应用程序定义了`.const`数据段，那么启动代码就会通过适当配置`PSVPAG`和`CORCON`寄存器将它们映射到程序空间的可视窗口内。

（3）实施数据初始化（仅有`crt0.o`进行）。

（4）调用应用程序入口函数`main()`，启动用户应用。 [267]

请注意，Microchip C编译器包含的标准启动代码除了完成上述的寄存器初始化外，不进行任何硬件初始化，用户的应用程序代码才负责配置硬件以及需要的其他软件配置。

大多数时候，启动代码对应用系统程序员是透明的。它作为标准链接器的一部分，在复位向量地址单元放置一个GOTO命令就可以将程序跳转至启动代码，而其他代码则被链接到标准启动代码模块`crt0.o`中，待启动代码完成所需的C语言运行环境初始化后，再调用用户的程序。然而，应用程序偶尔需要在系统启动后立即配置某些关键的系统资源，这时用户可以修改汇编语言文件`crt0.s`和`crt1.s`，以完成适当的修改。如果需要修改，那么程序员必须在项目源程序列表中包含文件`crtx.s`（其中x可以是0或者1），以确保它们被正确地链接到最终的应用程序中。 [268]

---

① 参见 *MPLAB C30 C Compiler User's Guide* 一书的 Startup and Initialization 一节。

附录C 带缓冲和中断驱动的串行I/O

在很多有关串行通信的论述中，作者都假设读者很熟悉可靠的发送和接收数据例程，因此是在比较高的层次谈论该问题。尽管书中经常将这些通信程序作为练习留给读者自己研究，但是开发一个可靠的底层通信接口程序的问题通常会因处理器的不同而有明显变化，甚至同一家公司的处理器也不完全一样。实际中，固化串行通信程序的关键是创建一个带缓冲和中断驱动的I/O架构。下面就研究这个术语的确切含义。

*I/O架构*是我们用于实施各种输入（I）和输出（O）串行通信的基本结构。与该架构一起，我们应确保所设计的设备能接收和发送各种需要处理的数据。

*中断驱动架构*是指通信硬件能够向处理器发出中断以便处理器能立即处理新的通信事件，然后再返回它之前正在处理的任务。如果打算使用这种架构，那么就得采用轮询架构，让处理器周期性地查询通信硬件，看看是否有事情需要处理。我们设计的应用程序框架要保证处理器能及时完成所有其他任务，以便它能查询出可能的最快的通信事件。鉴于通信事件发生得很快很频繁，因此，轮询架构只适用于最慢的通信信道。

最后，*带缓冲和中断驱动的架构*是指应用程序和通信信道之间传输的数据可以存储在队列中，而基本的中断服务程序和应用程序都能访问该队列，以确保队列中的数据是*一致的*（注意保证中断服务程序或者应用程序访问队列时不会发生冲突）。该方法为应用程序提供了最大的灵活性，能够确保实现极为可靠的通信，且处理开销最小。

这种架构的核心是数据队列，实际中它包括接收数据队列（用于保存设备已接收到来自其他系统的数据）和发送数据队列（用于保存设备将向其他系统发送的数据）。显然，需要设备处理的数据量远超出了任何实际处理器有限的存储器空间，因此数据队列通常被设计成*循环缓冲器*，数据会被依次加入缓冲器，当添加到缓冲器末尾后，又开始从缓冲器的起始位置添加新数据。数据删除过程也与此类似。习惯上，即将添加数据的位置被称为缓冲器的*头*，而即将删除数据的位置被称为*尾*。

使用该队列时，应用程序级的函数中断服务程序的函数都包含两组索引来确定即将访问的缓冲器的位置。每个索引的目的取决于使用索引的函数执行等级（应用程序等级或者中断服务程序等级），而队列则取决于运行哪个函数。比如，当应用程序希望向其他系统发送数据时，应用程序级的函数就会将数据写入发送数据队列，并且更新指针，以便应用程序能够将写入下一个需要发送的数据。当处理数据发送的中断服务程序发现有新数据需要发送时，它就会读取应用程序之前写入的数据，并更新中断服务程序的指针到下一个可以从队列读取数据的位置，然后将数据发送至通信硬件。此时，应用程序级函数负责更新环形发送数据缓冲器的头指针，而中断服务程序函数则负责更新缓冲器的尾指针。接收数据缓冲器的情况与此恰好相反，它的中断服务程序负责更新缓冲器的头指针，而应用程序级代码则负责更新尾指针。

这种方案在大多数情况下都能正常运行，除非某个或者所有队列变空。这时，头指针就会赶上尾指针，这种情况被称为*缓冲器溢出*。如果允许缓冲器溢出，那么位于尾指针和溢出

272 的头指针之间的所有数据都将丢失。由于在上述情况下，数据无论如何总会丢失，因此这种实现会在缓冲器变满后停止接收数据，而不是丢弃已接收到的数据。请注意一旦缓冲器空出一个或者多个字符空间，那么它又会继续接收数据。

C.1 架构的伪代码

串行结构的实际代码位于CommIF.c和CommIFDef.h文件中。这里介绍的伪代码主要是从高级别描述通信接口要执行的任务。为了使用该接口，应用程序必须首先初始化通信模块的硬件以及相关的全局状态变量，并启用相关的处理器中断。之后，应用程序就能利用下面介绍的函数从串行端口读取数据或者向串行端口写入数据。

C.2 系统初始化

为了初始化通信系统，用户需要调用函数CommInit()，并且指定所使用的UART（1或2）的参数，以及其他必要的通信参数（比特率、校验类型以及停止位个数）。

调用顺序：

```
Uint16 CommInit(Uint8 ui8Port,
                Uint16 ui16BaudRate,
                Uint16 ui16Parity,
                Uint16 ui16StopBits)
```

例如：初始化UART1，通信比特率为19.2Kbit/s，无校验，1个停止位。

```
CommInit(UART_1, 19200, PARITY_NONE, STOP_BITS_1);
```

C.3 从通信接口读取数据

为了从通信接口读取数据，应用程序需要调用CommGetRxPending()函数，以确定是否有新数据可读，如果该函数的返回值（它表示新增数据的字节数）大于0，那么应用程序就会通过调用函数CommGetRxChar()函数从接收数据队列中读取一个数据，该函数使用的参数是一个指向8位地址的指针，该地址单元保存了从队列中读取的数据。如果CommGetRxChar()函数的返回值非零，那么就表明发生了错误（参见文件StatusDef.h，273 其中包含完整的状态码定义），应用程序就不会使用当前参数指向的缓冲器数据。

调用顺序：

```
Uint16 CommGetRxPendingCount(void);
Uint16 CommGetRxChar(Uint8 *pui8Data);
```

例如：检查来自通信接口的新增数据，并且将其读入缓冲器。

```
Uint8 ui8Data;    // Buffer to hold data from Rx queue

if (CommGetRxPendingCount() > 0)
    CommGetRxChar(&ui8Data);
```

C.4 向通信接口写入数据

为了向通信接口写入数据，应用程序要调用函数CommPutChar()发送一个字节数据或者调用函数CommPutBuff()发送一个缓存区的数据。无论是哪种情况，如果函数的返回值非零，那么就表明发生了错误（参见文件StatusDef.h，其中包含完整的状态码定义）。

调用顺序

```
Uint16 CommPutChar(Uint8 ui8Data);

Uint16 CommPutBuff(Uint8 *pui8Data, Uint16 ui16Length);
```

例如：发送字符“d”。

```
Uint8 ui8Data;       // Buffer to hold data for Tx queue


ui8Data = 'd';

CommPutChar(ui8Data);
```

例如：发送字符串“dsPIC”（没有NUL终止符）。

```
CommPutChar("dsPIC", 5);
```

274



索　　引

索引中的页码为英文原书页码，与本书中页边标注的页码一致。

E

F

N

O

P

R

致　　谢

本书的起源是我在2003年1月的*Sensor Magazine*上撰写的一篇题为*Digital Signal Processing Turns Thermocouples Into Superstars*的文章。这篇文章有幸被Elsevier出版社Butterworth-Heinemann/Newnes部门中当时担任策划编辑的Carol Lewis女士看中。她亲切地询问我是否能撰写一部基于dsPIC的智能传感器方面的书籍。尽管Carol随后不久就退休了，而那时本书的手稿还尚未完成，但是幸运的是，她还能修订最终的草稿。对于Carol对我写作水平的信任以及她随后在整理最终草稿时付出的努力，我深表感谢。

我还很幸运地遇到了我的新编辑Tiffany Gasbarrini，她就像是一位监护人、朋友和保护者，偶尔还像个驱赶者（这只在极其必要的情况下才是）。如果没有她以上述四种角色的帮助，本书可能永远也无法面世，她和我的妻子能为此证明。Tiffany还得到了Michele Cronin的有力协助，她在我完成手稿前6个月左右才加入这个团队。Michele的细心和活泼的个性为本书的技术质量做出了重要贡献，也为我的撰写工作带来了很多乐趣。

在过去的几年中，我有幸能与很多杰出的工程师一起工作，他们使我对工程设计，特别是硬实时系统的设计又有了新的认识。这其中有四位杰出代表，我已经从他们那里学到了很多，相信读者也可以从中获益。Dale DuVall、John Bateson和Dale Redford是我在20世纪80年代后期在Scantek公司相识的，他们教会我透彻理解任何电子系统的基础物理过程的价值。这三位是我合作过的最好的工程师，他们都有各自擅长的领域，并能在设计过程中提出不同的思路。

Dale DuVall是一位实验物理学家，他对任何事都追根究底的态度很是了不起。他拥有将不同研究领域的类似现象联系起来的非凡能力，从中我学会了要保持注意力集中并且不忽视异常现象的重要性。John Bateson绝对是我所认识的最好的模拟电路设计师，同时也是最聪明和最令人快乐的人之一。John为人细心，处事得当，是我眼中的典范。Dale Redford是一位卓越的数字电路设计师，总是在电路和系统设计中不达完美不罢休。在Scantek工作期间这三位特别的伙伴给我留下了深刻的印象。

第四位工程师也恰巧是我的一个亲密朋友，他是我在Omnisys公司长达11年的经营伙伴，并且还是一位曲棍球队友：Fred Frantz。我们曾经一起设计过用于产品生产的控制和通信系统，这些产品包括新生儿心脏监控器、汽车减震器以及豪华游艇的电子驾驶系统。在这些产品的设计过程中，我从Fred身上学到了很多。我们一起度过了非常愉快的时光。

当然，Omnisys公司的设计工作也不是只由两位设计师完成，在这里我要感谢Parry Admire、John Brashier以及Mary Frantz（对，她是Fred的妻子，而且是一位出色的工程师），还有Steve Gibson，感谢他们所带来的友谊和出色的工程技巧。

最后，我要向我的家庭致以最深的谢意。直到我自己有了孩子，我才认识到我的父母为我的成长付出了多大的精力和关爱，以及我的姐姐Susan和Sarah对我有多么的容忍。当Susan嫁给James Belote时，我发现自己有了一个兄弟，而之前我从来没有过。他（如今还有他们的孩子）总能给每一次家庭聚会带来欢乐。当我迎娶Lisa时，我的家庭又增添了一位伟大的岳父Johnie和一位伟大的岳母Gerhard Schulz，他们对我来说就是第二个父亲和母亲。迎娶Lisa是我今生至今做出的最明智的决定，我很感激能和她共度每一天。而我们的三个孩子，Kate、Beth还有Dan，他们真的给我们的生活带来了光彩，这也证明了我是多么地幸运。

Intelligent Sensor Design
Using the Microchip dsPIC

智能传感器设计

"想快速掌握智能传感器系统设计吗？读这本书吧，你一定不会失望。"
——*Electronic Design*

智能传感器是一种新兴的、正在蓬勃发展的电子设备。它可以实现高精度的信息采集，具有一定的编程自动化能力，功能多样并且成本低，已经广泛应用于医疗和工业生产等领域。

本书正是开发智能传感器系统的实战指南，以最流行的芯片dsPIC为示例，但主要内容适合各种开发平台。书中力求通过实践指导，引领读者迅速掌握智能传感器的基础知识和设计方法，轻松开展应用设计。内容紧紧围绕智能传感器系统设计的特点展开，避免讲述深奥的理论，更多从物理意义出发，深入浅出地为读者展示各种概念的直观含义，并用三个完整的设计实例帮助读者解决设计与实现中的诸多实际问题，具有很高的实战参考价值。

Creed Huddleston 资深传感器设计专家。现任Omnisys公司副总裁。在嵌入式系统设计行业工作长达20余年，具有丰富的实践经验。

推荐阅读

- 《传感器技术手册》 Jon S. Wilson 主编
- 《嵌入式系统设计的艺术》(第2版 · 英文版) Jack Ganssle 著
- 《嵌入式系统：硬件与软件架构》 Tammy Noergaard 著
- 《PIC嵌入式系统开发》 Tim Wilmshurst 著

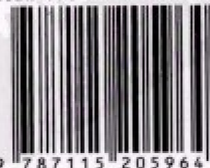
本书译自原版*Intelligent Sensor Design: Using the Microchip dsPIC*，并由Elsevier授权出版。





本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010) 51095186
反馈/投稿/推荐信箱：contact@turingbook.com

分类建议：电子电气/测试与测量
人民邮电出版社网址 www.ptpress.com.cn



ISBN 978-7-115-20596-4
9 787115 205964 >
ISBN 978-7-115-20596-4/TN
定价：45.00元